Knowledge-Based Systems 257 (2022) 109927

Contents lists available at ScienceDirect

Knowledge-Based Systems

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

Dynamic clustering based contextual combinatorial multi-armed bandit for online recommendation

Cairong Yan [a], Haixia Han [a,*], Yanting Zhang [a], Dandan Zhu [a], Yongquan Wan [b,c,**]

[a] Donghua University, Shanghai, China
[b] Shanghai University, Shanghai, China
[c] Shanghai Jian Qiao University, Shanghai, China

ARTICLE INFO

Article history:
Received 3 May 2022
Received in revised form 26 August 2022
Accepted 18 September 2022
Available online 23 September 2022

Keywords:
Online recommendation
Dynamic clustering
Contextual multi-armed bandit
Implicit feedback

ABSTRACT

Recommender systems still face a trade-off between exploring new items to maximize user satisfaction and exploiting those already interacted with to match user interests. This problem is widely recognized as the exploration/exploitation (EE) dilemma, and the multi-armed bandit (MAB) algorithm has proven to be an effective solution. As the scale of users and items in real-world application scenarios increases, their purchase interactions become sparser. Then three issues need to be investigated when building MAB-based recommender systems. First, large-scale users and sparse interactions increase the difficulty of user preference mining. Second, traditional bandits model items as arms and cannot deal with ever-growing items effectively. Third, widely used Bernoulli-based reward mechanisms only feedback 0 or 1, ignoring rich implicit feedback such as behaviors like click and add-to-cart. To address these problems, we propose an algorithm named Dynamic Clustering based Contextual Combinatorial Multi-Armed Bandits (DC^3MAB), which consists of three configurable key components. Specifically, a dynamic user clustering strategy enables different users in the same cluster to cooperate in estimating the expected rewards of arms. A dynamic item partitioning approach based on collaborative filtering significantly reduces the scale of arms and produces a recommendation list instead of one item to provide diversity. In addition, a multi-class reward mechanism based on fine-grained implicit feedback helps better capture user preferences. Extensive empirical experiments on three real-world datasets demonstrate the superiority of our proposed DC^3MAB over state-of-the-art bandits (On average, +75.8% in F1 and +54.3% in cumulative reward). The source code is available at https://github.com/HaixHan/DC3MAB.

1. Introduction

Personalized online recommendation plays an important role in alleviating information overload by customizing satisfactory items for target users [1]. It faces a trade-off between two goals [2]: exploring new items for improving user satisfaction in the long run while exploiting known information to recommend items that have already been interacted. This issue is rooted in the exploration and exploitation (EE) dilemma. Traditional recommendation algorithms, such as collaborative filtering, become ineffective because of their offline training paradigm. Multi-armed bandits (MABs) have received much attention and are widely used to deal with this problem [3–5].

Fig. 1 shows an example of modeling the personalized online recommendation task as a MAB problem. The agent selects an action sequentially from a finite set of actions (arms) with a fixed but unknown reward distribution and observes the feedback of the action to provide further references for the next action selection. In the exploration phase, it takes a new action; in the exploitation phase, it pulls the arm with the highest reward probability based on known information. After the agent takes action, a reward is generated, which helps the agent take the next action. This mechanism will maximize the cumulative reward over time or minimize the cumulative regret.

Although much progress has been made, some key issues still need to be investigated when we build MAB-based recommendation models. We list these challenges below.

First, the vast and growing number of users with different preferences increases the complexity of MAB modeling. Most MAB-based recommendation models fall into two extremes when creating bandits for users, building a global bandit for all users [6] and creating an independent bandit for each user [7]. The former is simple to update the global parameters and also helps learn from the observed reward information of all users. The latter provides recommendations to users in a completely personalized

* Corresponding author.
** Corresponding author at: Shanghai University, Shanghai, China.
E-mail addresses: cryan@dhu.edu.cn (C. Yan), haixiahan@mail.dhu.edu.cn (H. Han), ytzhang@dhu.edu.cn (Y. Zhang), ddzhu@dhu.edu.cn (D. Zhu), wanyq@gench.edu.cn (Y. Wan).

https://doi.org/10.1016/j.knosys.2022.109927

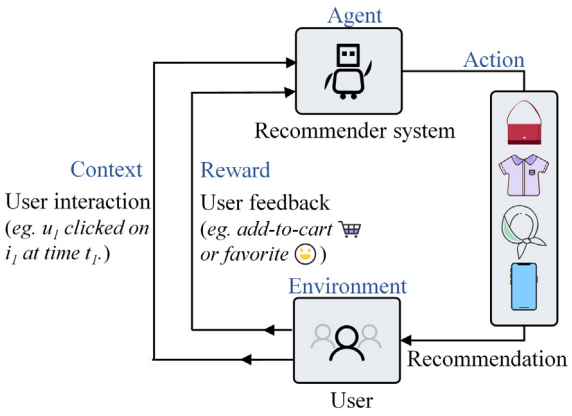

Fig. 1. An example of modeling the personalized online recommendation task as a MAB problem. Black labels represent concepts in the recommendation task, and blue labels represent concepts in the bandit problem.

style. Although user preferences are unique, some are similar, so customizing a bandit for a group of similar users rather than a single user or all users can take advantage of collaborative information and reduce overhead, and is an efficient solution [8]. It is worth noting that how to quickly and accurately cluster users in dynamic environments remains to be studied. In this paper, we propose a dynamic user clustering strategy that enables different users in the same cluster to estimate expected rewards for arms cooperatively.

Second, the ever-growing number of items leads to a large and non-fixed number of arms, increasing the computational complexity. The traditional MAB-based recommendation models treat each item as an arm [9]. The large-scale arms will lead to higher computational complexity because the algorithms need to calculate the reward distribution for each arm every time. Moreover, many algorithms recommend only one item at a time, which cannot meet the demand for various services, leading to low recommendation accuracy. Our previous research found that modeling item categories rather than items as arms can control the scale of arms to improve recommendation performance [10], but did not address the problem of item dynamic partitioning. So in this paper, we will investigate a collaborative filtering-based approach to dynamically partitioning items to reduce the number of arms and generate a recommendation list to improve diversity.

Third, as the number of users and items increases, their explicit interactions, such as ratings and purchases, become sparser, making capturing user preferences more challenging. Fortunately, the current recommender systems collect rich implicit feedback data that contains a large amount of information about users' preferences [11]. For example, users click on a recommended item or add it to their shopping carts, which means they are interested in the recommended items. Moreover, users may prefer items clicked twice or added to a shopping cart rather than items clicked only once. However, most existing related work uses a Bernoulli-based reward mechanism, i.e., setting the reward to 1 when the recommendation is booming and 0 otherwise. In this case, only the purchased recommended items are considered successful recommendations, and the implicit feedback plays no role in preference mining. Therefore, we will study various types of implicit feedback, classify them, observe their impact on the model, and design a multi-class reward mechanism based on fine-grained user implicit behavior to help better capture user preferences.

To address the above challenges, we propose a recommendation algorithm based on a contextual multi-armed bandit (CMAB), an extensively-studied variant of MAB [12–14]. In contrast to MAB, CMAB leverages the contextual information contained in the environment. For example, in an e-commerce recommendation platform, user attributes such as gender, age, income, previous likes, item attributes such as category, color, brand, or even the reward probability of each arm can all be contextual information. These auxiliary data will help recommender systems to make more personalized and high-quality recommendations. We summarize the contributions as follows:

- We first formalize the online recommendation task as a contextual combinatorial multi-armed bandit problem and give definitions of key components in the bandit, respectively.
- We propose a Dynamic Clustering based Contextual Combinatorial Multi-Armed Bandit (DC^3MAB) algorithm, including a dynamic user clustering strategy, a collaborative filtering-based dynamic item partitioning approach, and a multi-class reward mechanism.
- Extensive experiments show that compared with the state-of-the-art algorithms, the proposed DC^3MAB improves the recommendation performance metric F1 by 12.6%, 153.1%, and 61.6% on the IJCAI-15, Retailrocket, and Yoochoose datasets, respectively. We also publicly release the source code.

The remainder of this paper is organized as follows. We first briefly review the related works on MAB-based recommender systems in Section 2. We formulate the online recommendation task as a contextual combinatorial bandit problem and present the DC^3MAB algorithm in Section 3. Section 4 shows the experimental results. Finally, Section 5 concludes the paper.

2. Related work

This paper mainly focuses on several problems encountered when applying MAB for the recommendation tasks. So we briefly review related bandit algorithms from the following four aspects.

Contextual MAB (CMAB). The CMAB problem has been a hot research topic in statistics and machine learning. In CMAB, suppose that $\mathcal{A}$ represents a finite set of arms and each arm $a \in \mathcal{A}$ is represented by a d-dimension context vector. The context describes the observable information of arm a and the target user at time t. The agent chooses an arm a^* for the target user and then receives a reward $r_{t,a^*} \in \mathcal{X}$, where $\mathcal{X} = \{0, 1\}$ represents a binary reward [15]. LinRel [3] is a well-known algorithm that pioneers the idea of the CMAB problem with linear payoffs. It assumes the reward is a linear combination of the arm's previous rewards. Like this idea, LinUCB [6] is one of the most popular linear random algorithms. It sequentially selects items to serve users based on the contextual information about users and items and adjusts its item selection strategy based on user click feedback to maximize the total user clicks. It has been applied to Yahoo! Front Page Today Module.

Xu et al. [16] addressed the challenge of time-varying interests by employing a change-detection procedure to identify potential changes on the preference vectors. Additionally, Agrawal et al. [17] attempted to generalize Thompson sampling (TS) to the stochastic contextual MAB problem with linear payoff functions. So far, CMAB has been applied in various recommendation fields to capture user preferences and provide personalized recommendations, such as e-commerce recommendation [18], news advertisement recommendation [19], music and movie recommendation [20], and article recommendation [21].

Combinatorial MAB. Whether in contextual or non-contextual bandit algorithms, each arm is usually modeled as an item, i.e., pulling an arm once will recommend one item corresponding to that arm to the target user. However, in real-world scenarios, recommender systems always provide each user with a set of

items rather than one item at a time [22,23]. The approach of recommending only one item at a time is not only difficult to satisfy the diverse needs of users but also difficult to accurately match the user's preference. Therefore, the multi-arm selection problem has become a research focus. The method of ranked bandits [7] generates multiple candidate items through multiple bandits, where each bandit is independent, but it does not take full advantage of the available feedback information. The cascading bandits [24,25] recommend an ordered list of items to the user, and the user stops on the first satisfied item when checking the item lists. Based on this work, Zong et al. [26] considered a set of linear cascading bandits to deal with an ever-changing set of items. Some algorithms extend the traditional algorithms of selecting a single action at a time to selecting multiple actions. Komiyama et al. [27] proposed a combinatorial multi-bandit containing a constraint $\mathcal{A} \in 2^{|I|}$, where I represents the total items, and $2^{|I|}$ is the set of all possible subset of items. The MP-TS algorithm [27] reassembles the classic Thompson sampling algorithm, which selects the top N best arms from a pool of candidates instead of selecting the best arm based on posterior samples. There are also some studies [28,29] that generalize the classical non-contextual bandit to combinatorial bandit. In a combinatorial multi-armed bandit, the learner can play a set of arms, called super arm, which is a set of basic arms that obey a specific distribution. However, these efforts do not exploit any contextual information. Therefore, much work must be done to make the bandit algorithms more useful in recommendation tasks.

Clustering-based bandits. Some studies incorporate the clustering idea into bandit algorithms to reduce recommendation difficulty and maintain a promising regret constraint. They perform clustering operations from different perspectives, including bandits, arms, sessions, and users. Gentile et al. [30] focused on clustering arms (items) based on some dependencies. The CLUB method [31] clusters online linear bandits by maintaining a graph among users and using connected components to represent user clusters. The DynUCB method [8] clusters users into k groups so that parameters are shared among the bandits of various users in the same cluster to help obtain better arms. Both CLUB and DynUCB implement dynamic user clustering. The difference is that CLUB continuously deletes users from the clusters, while DynUCB dynamically adjusts the target user's cluster. The COFIBA [23] considers the combined effect, i.e., some users respond similarly to the recommended items, so that users are dynamically grouped according to the items interacted. The ClexB [32] designs an adaptive clustering strategy to share knowledge among users, and it also enhances collaborative filtering work via exploring users clustering. Sanz-Cruzado et al. [33] developed a simple user clustering method, which can be viewed as a variant of the nearest neighbor scheme. The method provides users with a controlled ability to explore similar users randomly. However, it needs to compare the target user with other users at each round and then re-perform the clustering operation to find similar users, which is too time-consuming.

Modeling implicit feedback. Most users do not directly tell systems how satisfied they are with the recommended items, so collecting explicit feedback in ratings or stars is very costly, and the feedback collected is usually very sparse. In contrast, recommender systems can easily collect implicit feedback data that provide a more comprehensive picture of user preferences [11]. The most common way to deal with implicit data [34] is to convert them directly to 0 or 1. Cascading bandit [35] is a straightforward but effective way to characterize user behavior. However, it considers only one type of behavior and ignores the diversity of behaviors. More Recent related work has focused more on differential behavior types of implicit feedback. Schoinas et al. [36] created a denser user-item matrix based on association rule mining (ARM), where the association rules are used to extract and fill in missing values based on behavior history. They set the user-item feedback to 1 for those with interaction and 0 for those without interaction and output the final prediction score by weighted summation of multiple actions and confidence intervals. Gao et al. [37] provided a solution called neural multi-task recommendation (NMTR) for learning preferences from users' multi-behavior data. They employ a neural network model to capture complex, multi-type interactions between users and items and jointly optimize them based on a multi-task learning framework to exploit feedback signals.

All of the above work can improve the recommendation performance. However, how effectively to combine these into the recommendation framework is a technical difficulty. This paper will conduct an in-depth exploration to propose a flexible framework.

3. Methodology

In this section, we present our proposed recommendation framework. We first define our problem and then discuss an efficient solution. Finally, we summarize the key steps into an algorithm.

3.1. Problem formulation

Following the reinforcement learning paradigm, we first model the typical e-commerce recommendation task as a contextual combinatorial bandit problem. At the time t, the recommender system observes the user interaction (context) and the user feedback (reward) it has collected from previous rounds, takes a recommendation strategy (action) according to a policy function, and provides the target user with a recommendation list including multiple items. The recommender system receives new user feedback (reward) corresponding to the user's response. The user response reflects feedback on the quality of the recommended items and is used to update the parameters of the recommender system to help make more accurate actions to provide better recommendations in the next round. Fig. 1 shows the workflow of a typical personalized online recommendation task and the architecture of the bandit problem. The combination of the two shows the potential advantages of leveraging bandit for recommendation tasks. Several critical components in the bandit are defined below.

Agent. The core problem of reinforcement learning is how to maximize the reward that an agent can obtain in a complex and uncertain environment. Likewise, recommender systems aim to maximize user satisfaction.

Environment. Suppose the item set I, the user set U, and the behavior sequence set of interaction behavior S between them. The behavior sequence of user u is expressed as $\hat{S}_u^T = (S_{u1}, S_{u2}, \ldots, S_{uT})$, where T represents the end time in the sequence and S_{ut} represents a record of interaction behavior of user u at time t ($1 \leq t \leq T$). The record consists of user ID, item ID, attributes, and the interaction type.

Context. In many bandit problems, the agent has access to additional information that may help predict the quality of the actions, called context. Similarly, in recommendation field, the auxiliary information about users or items is used to capture user preferences, thereby improving the quality of personalized recommendations. The context information for the system is represented as c_t at time t.

Action. In the contextual combinatorial bandit, there are J available arms $\mathcal{A} = \{a_1, a_2, \ldots, a_J\}$ that can be pulled by user u at time t, where $|\mathcal{A}| \ll |I|$. An arm corresponds to a subset of

items instead of a single item, called a super arm. Each super arm has a chance to be pulled. For each super arm $a \in \mathcal{A}$, the bandit observes a context c_t which is in the form of a d-dimensional contextual feature vector $x_{t,a} \in R^d$. Based on the reward experience in previous iterations, the agent chooses to pull an optimal super arm a^* according to

$$a^* \sim \pi_\theta(\cdot \mid c_t) = \text{argmax}_{a \in A} \mathbb{E}(R_a) + f^a_{UCB}, \tag{1}$$

where $\mathbb{E}(R_a)$ is the estimated reward of the super arm a, determined by bandit parameters, and f^a_{ucB} is the function used to calculate the upper confidence bound of super arm a. Each time the bandit chooses the super arm with the highest upper confidence bound. Then the online contextual combinatorial bandit-based recommender system generates a recommendation list $\bar{I}_{ut}(|\bar{I}_{ut}| \geq 1)$ corresponding to a^* for user u at time t.

Reward. After a^* is pulled, the agent observes a reward r_{t,a^*}. The reward refers to users' responses or feedback to the recommended items in the recommendation field. Then agent learns from the observations of $(x_{t,a^*}, a^*, r_{t,a^*})$ in the ongoing iteration to improve its strategy for choosing the best super arm in future iterations. After T rounds, the agent observes a cumulative reward of $\mathbb{E}\left[\sum_{t=1}^{T} \mathrm{R}_t^*\right]$, where R_t^* is the expected reward obtained by the selected super arm at time t and the cumulative regret (abbreviated as *Reg*) over T rounds is defined formally by

$$\text{Reg(T)} = \mathbb{E}\left[\sum_{t=1}^{T} \max_{a \in A} \mu_a\right] - \mathbb{E}\left[\sum_{t=1}^{T} R_t^*\right], \tag{2}$$

where the first term is the maximum expected reward using any super arm, i.e., $\mathbb{E}\left[\sum_{t=1}^{T} \max_{a \in A} \mu_a\right] \geq \mathbb{E}\left[\sum_{t=1}^{T} R_t^*\right]$, and μ_a represents the expected reward of the super arm a.

Reg (T) is usually used to measure the performance of bandit algorithms, and a smaller value indicates a better result. It is unlikely to observe the payoff of every arm. Thus it is hard to calculate Reg (T). The total regret minimization is equivalent to the total reward maximization. So, the goal of contextual combinatorial bandit learning is formally to maximize the cumulative reward after T rounds, i.e., $\max\left(\mathbb{E}\left[\sum_{t=1}^{T} R_t^*\right]\right)$.

3.2. Recommender system based on DC^{3}MAB

Designing fast and accurate recommendation algorithms based on contextual information has always been a key issue in online recommender systems. Fig. 2 shows the structure of a recommender system based on *DC^{3}MAB*, which consists of four main modules: (a) user clustering, (b) item partitioning, (c) bandit, and (d) output. Module (a) computes the similarity of users and clusters them. The input is the sequence set of interaction behaviors. Module (b) is responsible for partitioning items into different subsets, which will act as the super arms of the bandit. Module (c) generates a candidate recommendation list for user u via a bandit policy. Module (d) outputs the final recommended items and provides feedback and rewards for Module (a) and Module (b). Next, we will describe these modules in detail. We will describe the core functions involved in these modules.

Dynamic user clustering. To achieve dynamic clustering on the user side, each user cluster C_j maintains a cluster parameter $\bar{w}_{C_j}$ and each user u has a unique parameter w_u. Initially, the k-nearest neighbors (KNN) algorithm is used to cluster all users based on user feature vectors. It is worth noting that we only use the KNN algorithm to cluster users at the beginning. Later, a different strategy is used to dynamically adjust the clusters of users, as the KNN algorithm is time-consuming and not suitable for online real-time recommendations. How to dynamically adjusts the target user's cluster will be described in detail below.

Suppose the recommender system serves user u at time t, and the input is the interaction information of u to infer u's preference. The orange circles of Module (a) in Fig. 2 represent multiple user clusters. The small red box represents the target user u currently being served by the system.

Through user clustering, the recommender system finds similar users to the target user and utilizes the collaboration of similar users to determine the recommendation list. This will solve the problem of insufficient information available due to sparse data. The strategy also leverages the wisdom of similar users to make decisions.

Dynamic item partitioning. Assume that there are J different predictors ($\mathrm{J} > 0$ and $\mathrm{J} \ll |\mathrm{I}|$) to divide the total items according to S_{ut}, each predictor can use a different collaborative filtering technique. We distinguish predictors into two types: attribute-based predictors and behavior-based predictors. Attribute-based predictors select items with similar attributes to the interacted items, e.g., they retrieve items of the same color or brand as the interacted items or the most popular items in the same age group. In general, according to the characteristics of the dataset, the attributes of users or items (movies, articles, music, products) can be used to filter the total items so that items with the same properties as the interacted items can be grouped into a subset of items, which will correspond to a super arm. Behavior-based predictors select items via events or interaction behavior. For example, we define users with the same interaction behavior as the target user's current interaction with similar users. These similar users' favorite items constitute a subset of items.

In addition, the user preferences may change, and even the same user tends to be interested in different items at different times. The items included in each super arm also change dynamically according to user behavior.

Bandit. By observing the context information of each super arm, the algorithm chooses the one that maximizes the upper confidence bound as the selected super arm a^*.

$$a^* = \text{argmax}_{a \in A}\left(\bar{w}_{C^*}^T x_{t,a} + CB_t\right), \tag{3}$$

$$CB_t = \alpha\sqrt{x_{t,a}^T \bar{M}_{C^*}^{-1} x_{t,a} \log(1+t)}, \tag{4}$$

where CB_t is a simplified version of the theoretical upper confidence bound, $\bar{w}_{C^*}$ is the parameter of the cluster in which user u is located, and it reflects the preference of cluster C^*. Here, α is the parameter for the importance of exploration.

Once the super arm a^* is selected, the item subset corresponding to it will be used as the candidate recommended items. In the real world, not all super arms are available each time. It happens when predictors find no correlations between items. Our bandit policy is capable of dealing with such situations effectively. Usually, the length of the selected super arm a^* will be greater than N ($|a^*| >= \mathrm{N}$), so the algorithm will randomly choose N items from it. When it is less than N, all items included in the super arm are the final recommended items. When it is 0, the algorithm will skip this selected super arm and pull a new super arm again. Moreover, to ensure meaningful recommendations, it performs a filtering operation on the selected super arm a^* according to preset rules. For example, it removes the already purchased and duplicate items in a^*.

Multi-class reward mechanism. $\bar{I}_{ut}$ represents the final recommendation list for user u at time t. Unlike the method of using a loss function to optimize in machine learning, the bandit algorithm is based on the previous reward feedback to improve the next action continuously. Observing whether the user is interested in the recommended items is necessary when we verify the recommendation performance at time t. If the user does not purchase the recommended items at time $t+1$, we cannot directly

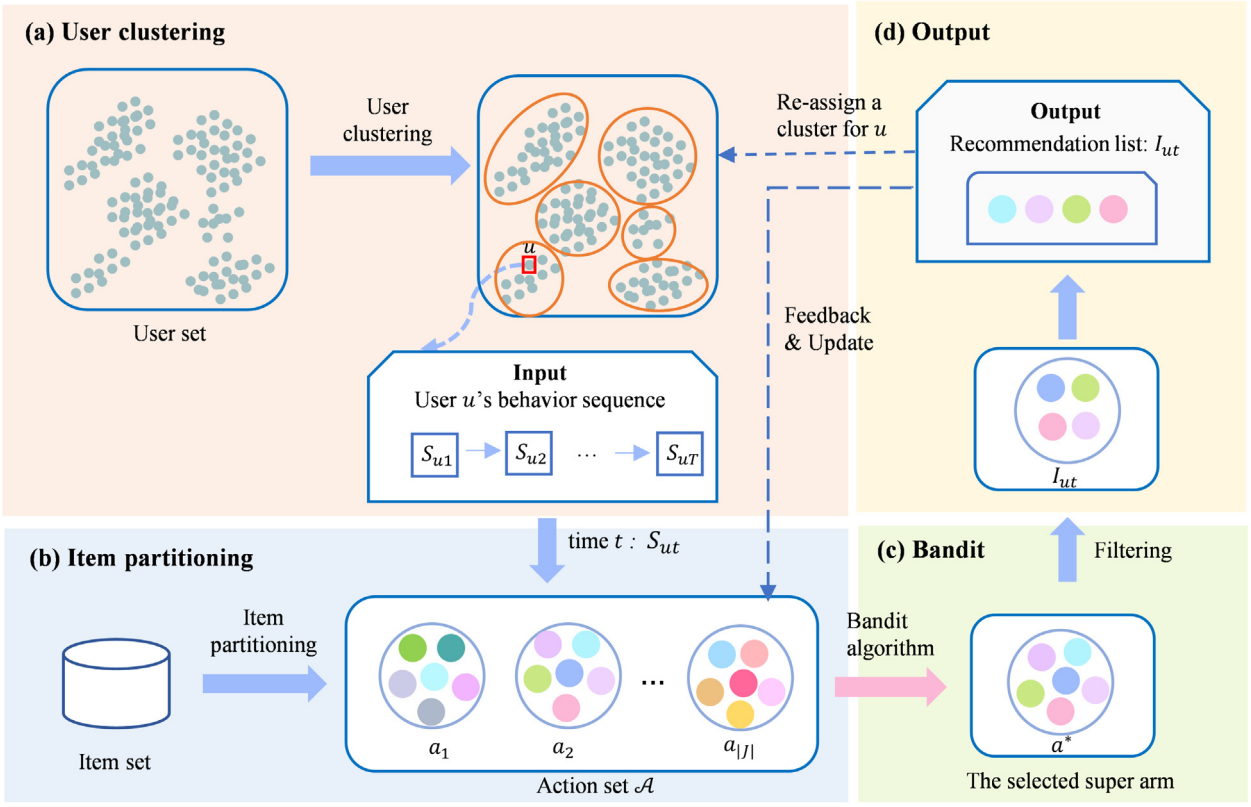

Fig. 2. The structure of the recommender system based on *DC*^{3}*MAB*. It consists of four main modules: (*a*) user clustering, (*b*) item partitioning, (*c*) bandit, and (*d*) output.

conclude that this recommendation fails. Here, we propose a concept of *strong preference*, mainly used to indicate that the user is interested in the item. For example, if a user clicks on the buttons of favorite, add-to-cart, or purchase for an item, we consider that the user is very interested in that item, and these three behaviors are called strong preference behavior. Otherwise, if a user clicks an item only once, we think the user is not very interested in the item, and we call it weak preference behavior. Most current work uses the Bernoulli-based reward mechanism, where the reward for a successful recommendation is 1 and 0 otherwise. However, in real applications, user behavior is complex and diverse. Since the Bernoulli-based reward setting does not utilize implicit feedback information and the implicit feedback can reveal the target user's preference, we propose a new reward mechanism called *Multi-class Reward Mechanism* (MC reward), in which the system will receive different feedback values for different behavior types as rewards. We set different reward weights for different interaction types to better describe the degree of preference. Suppose there are five types of behavior, including non-interaction, click, favorite, add-to-cart, and purchase. The reward weights for the above behavior types gradually increase according to the degree of preference revealed. Furthermore, the reward of item i is defined by:

$$r_i = \begin{cases} RW_{ui}, & \text{if it's a successful recommendation,} \\ 0, & \text{otherwise,} \end{cases} \tag{5}$$

where RW_{ui} represents the reward weight of item i when recommending item i to u. Further, the reward for the selected super arm a^* is defined as

$$r_{t,a^*} = \sum_{i \in \tilde{I}_{ut}} r_i, \tag{6}$$

After each round, the recommender system detects changes in user u's preference, and the related parameters of u need to be updated. According to the preference similarity between u and each user cluster, the recommender system will find the most suitable cluster for u.

3.3. *DC*^{3}*MAB algorithm*

The *DC*^{3}*MAB* algorithm is described in Algorithm 1. User preferences are associated with user clusters. Specifically, user set U is divided into k clusters $C_1, C_2, \ldots, C_k$, where $k \ll |U|$. Users in the same clusters have similar preferences, while users in different clusters have different preferences. When the algorithm serves user u, several predictors partition the total items into different subsets of items, modeling them as super arms $\mathcal{A} = \{a_1, a_2, \ldots, a_J\}$. The *getActions* function in Algorithm 1 is to implement the item partitioning function. The super arms are represented as a set of contextual feature vectors $\{x_{t,a_1}, x_{t,a_2}, \ldots, x_{t,a_J}\} \subseteq R^d$ at time t. Because the contextual information changes over time, the *getActions* function needs to be called every moment. The parameter of cluster C* where u is located is used to calculate the upper confidence bound. The algorithm selects a super arm a^* based on the policy function shown on line 11. That means a subset of items corresponding to the super arm a^* is recommended to the user u, and then a reward r_{t,a^*} will be observed. Algorithm 2 shows a detailed description of the MC reward mechanism. The contextual feature vector x_{t,a^*} and the reward r_{t,a^*} corresponding to the selected super arm are used to update the parameter of user u, which helps to adjust the user cluster where u is located. They are also used to update the reward distribution of the super arms, which provides help for the next round of decision-making. Specially, the algorithm uses contextual feature vector x_{t,a^*} of the selected super arm for updating w_u, which reflects the preference of user u and is defined by $w_u = M_u^{-1} b_u$. Note that updating the user parameter is only performed at w_u. Parameters

Algorithm 1: DC^3MAB

Initialize the k user clusters, and $b_{u'} = 0 \in R^d$, $M_{u'} = I \in R^{d\times d}$ for all users $u' \in U$
Compute the coefficient $\bar{w}_{C_j}$ for each cluster C_j:
$\bar{M}_{C_j} = I + \sum_{u'\in C_j}(M_{u'} - I)$
$\bar{b}_{C_j} = \sum_{u'\in C_j} b_{u'}$
$\bar{w}_{C_j} = \bar{M}_{C_j}^{-1}\bar{b}_{C_j}$
Select a user u to serve
for $t \leftarrow 1, 2, 3, \cdots, \frac{2}{3}T$ **do**
  Obtain u's user cluster C^*
  Define super arms at time t, $\mathcal{A} =$ **getActions**(S_{ut}, t)
  Observe the contexts of super arms. $x_{t,a} = \frac{1}{|a|}\sum_{a'\in a} x_{a'}, \forall a \in \mathcal{A}$
  Call bandit policy and choose an action, $a^* = \text{argmax}_{a\in\mathcal{A}}\left(\bar{w}_{C^*}^T x_{t,a} + \alpha\sqrt{x_{t,a}^T \bar{M}_{C^*}^{-1} x_{t,a}\log(1+t)}\right)$
  Get the output result $\bar{I}_{ut}$ corresponding to the selected super arm a^*
  Observe the reward through multi-class reward mechanism, $r_{t,a^*} =$ **getReward**$(\bar{I}_{ut}, t)$
  Set $\tilde{x}_t = x_{t,a^*}$
  Update u's parameters, and set:
  $M_u = M_u + \tilde{x}_t\tilde{x}_t^T$
  $b_u = b_u + r_{t,a^*}\tilde{x}_t$
  $w_u = M_u^{-1}b_u$
  Adjust the user cluster where u belongs to
  $\bar{C} = \text{argmin}_{i=1,\ldots,k}\|w_u - \bar{w}_i\|$
  if $\bar{C} \neq C^*$ **then**
    Move u from cluster C^* to $\bar{C}$
    Re-compute $\bar{w}_{\bar{C}}$
  end
  Re-compute $\bar{w}_{C^*}$
end

Algorithm 2: Function *getReward* $(\bar{I}_{ut}, t)$

Initialize the reward of selected super arm $r = 0$ **for** i *in* $\bar{I}_{ut}$ **do**
  if i $\in$ StrongPreferenceSet **then**
    $r \leftarrow r + \text{RW}_{ui}$
  end
end
return r

for users other than user u do not need to be updated. Once w_u changes, the bandit algorithm re-assigns user u to a cluster whose preference coefficient $\bar{w}_{\bar{C}}$ is closest to w_u. Then it re-computes $\bar{w}_{C^*}$ and $\bar{w}_{\bar{C}}$. Therefore, dynamic user clustering is achieved by finding the preference parameter of a cluster most similar to the target user, enabling the algorithm to adapt to changes in user preferences.

Below is the time complexity analysis of the algorithm. Initially, clustering all users takes $O(d|U|)$. At each step, item partitioning takes $O(J|I|)$ and each recommendation takes $O(Jd^2)$, where the matrix inverse is updated by the Sherman–Morrison formula. The time it takes to obtain a reward using the multi-class reward mechanism is $O(|S_u|)$. Updating user parameters takes $O(d^2)$ and updating cluster parameters takes $O(|U|/k)$. So, the computational time complexity for T rounds of DC^3MAB is $O(d|U|+T(J|I|+Jd^2+|S_u|+d^2+k+|U|/k))$. Here, $|S_u|$ is the average user sequence length, and $|U|/k$ is the average user number of the cluster.

Table 1
The statistics of datasets.

Dataset	Item	Attribute	Instance	Interaction	Length
IJCAI-15	8,348	5	15,861	5	16
Retailrocket	4,822	2	13,748	4	14
Yoochoose	2,619	2	13,602	3	14

4. Experimental results and evaluation

In this section, we conduct extensive experiments to answer the following questions:

- RQ1: How does DC^3MAB perform compared to the baselines for the online recommendation task?
- RQ2: How do the critical components in DC^3MAB, such as item partitioning and multi-class reward mechanism, affect the recommendation performance?
- RQ3: How do essential parameters such as feature dimension, user cluster number, and sequence length affect the performance of DC^3MAB?

4.1. Experimental settings

4.1.1. Datasets

We evaluate all the models on three datasets, IJCAI-15[1], Retailrocket[2], and Yoochoose[3], which are commonly used in the field of e-commerce recommendation. Each instance in the datasets consists of user ID, item ID, user and item attributes, interaction type, and interaction time. There are five interaction types in the IJCAI-15 dataset: non-interaction, click, favorite, add-to-cart, and purchase. There are four interaction types in the Retailrocket dataset, excluding favorite. For the Yoochoose dataset, there are only three interaction types: non-interaction, click, and purchase. To eliminate the influence of irrelevant data, we choose offline user behavior data with context information not empty to simulate online recommendations. Table 1 shows the details of the three datasets after preprocessing.

4.1.2. Evaluation metrics

We employ cumulative reward (CR), precision, recall, and F1-measure (F1) to evaluate the algorithms.

CR. It is a typical metric used to evaluate the performance of the bandit algorithm, defined as the sum of the rewards obtained by the algorithm in T rounds. The larger the cumulative reward, the closer the algorithm is to the optimal policy.

Precision. It is a typical metric used to measure the recommendation accuracy, defined as

$$\text{Precision}_u = \frac{1}{|\hat{S}_u^T|}\sum_{t=1}^{|\hat{S}_u^T|}\frac{|\bar{I}_{ut}\cap\bar{S}_u|}{|\bar{I}_{ut}|}, \tag{7}$$

where $\bar{S}_u$ represents the item list with a strong preference in the behavior sequence of user u. As already defined in the previous content, $\bar{I}_{ut}$ is the recommended item list, $\hat{S}_u^T$ is the behavior

[1] https://tianchi.aliyun.com/dataset/dataDetail?dataId=47.
[2] https://www.kaggle.com/datasets/retailrocket/ecommerce-dataset.
[3] http://2015.recsyschallenge.com/challenge.html.

Table 2
A simple example about user behavior sequence.

Time	t_1	t_2	t_3	t_4	t_5	t_6	t_7
Item	001	002	001	003	006	004	005
Behavior	2	3	3	1	1	1	2

sequence of user u, and T is the end time in the sequence. Then, the average precision of all test users is defined as

$$\text{Precision} = \frac{1}{|U|}\sum_{u\in U}\text{Precision}_u, \tag{8}$$

where $|U|$ represents the total number of users.

Recall. It is also a typical metric for evaluating recommendation performance, defined as

$$\text{Recall}_u = \frac{1}{|\hat{S}_u^T|}\sum_{t=1}^{|\hat{S}_u^T|}\frac{\left|\bar{I}_{ut}\cap\bar{S}_u\right|}{\left|\bar{S}_u\right|}. \tag{9}$$

Then, the average recall of all test users is defined as

$$\text{Recall} = \frac{1}{|U|}\sum_{u\in U}\text{Recall}_u\,. \tag{10}$$

F1-measure. F_β is a configurable metric used to evaluate binary classification models based on their predictions for positive classes. Here, it is defined as

$$F_\beta = \frac{\left(\beta^2+1\right)\times\ \text{Precision}\ \times\ \text{Recall}}{\beta^2\times(\ \text{Precision}\ +\ \text{Recall}\ )}. \tag{11}$$

When β =1, we call it F1-measure (F1). Here, we use F1 to evaluate the performance of different algorithms regarding the trading-off between precision and recall. $F1_u$ represents the $F1$ value of user u.

At time t, the above metrics of user u are denoted as $precision_u^t$, $recall_u^t$, and $F1_u^t$. Table 2 shows an example of behavior records of user u, where 0, 1, 2, 3, and 4 represent the interaction types of non-interaction, click, favorite, add-to-cart, and purchase, respectively. At t_1, the recommendation list returned by the recommendation algorithm is (001,004,002,002,010,015). Then, according to the above formulas, we can get the precision, recall and F1 for user u at time t_1: $precision_u^{t_1} = 3/5$, $recall_u^{t_1} = 1/2$, and $F1_u^{t_1} = 6/11$.

4.1.3. Feature construction

Constructing contextual feature vectors is an important step in CMAB. Matrix factorization (MF) is a simple yet effective embedding-based model for collaborative filtering [38]. we adopt this method to extract user and item features.

The process of extracting features based on MF is explained below. Suppose a user-item interaction matrix $\mathcal{R}\in\mathbb{R}^{|U|\times|I|}$, $|U|$ and $|I|$ denote the number of users and items respectively. MF maps both users and items into a joint latent feature space of d dimension such that interactions are modeled as inner products in that space. Vector p_u denotes the latent feature vector for u, similar notations for q_i. Matrix $P\in\mathbb{R}^{|U|\times d}$ and $\mathcal{Q}\in\mathbb{R}^{|I|\times d}$ denote the latent matrix for users and items. Mathematically, each user-item rating score $\hat{r}_{ui}$ in $\mathcal{R}$ can be estimated as

$$\hat{r}_{ui} = \langle p_u, q_i\rangle = p_u^T q_i. \tag{12}$$

The loss function is defined as

$$Loss = \sum_{r_{ui}\neq 0}\left(r_{ui}-\hat{r}_{ui}\right)^2, \tag{13}$$

where $\hat{r}_{ui}$ is the real rating score. In MF, the loss can be minimized using gradient descent or other methods to obtain the optimal P and Q.

Building an MF model requires a rating matrix $\mathcal{R}$ (whether it is sparse or not). However, as mentioned above, in most practical recommendation scenarios, there is little explicit feedback or ratings from users and only implicit user behavior on items, such as click and add-to-cart. To construct a rating matrix, we assign different rating scores to different interaction types, consistent with the assignment of reward weights RW_{ui}. In our experiment, we set the rating scores corresponding to click, favorite, add-to-cart, and purchase as 1, 2, 3, and 4, respectively. If there is no interaction, the rating score is set to 0. If the user has multiple different types of interactions with the same item, we use the maximum as the final score.

After the feature vectors of users and items are obtained, we calculate the feature vector of each arm. For a super arm a, we use the average of the feature vectors of all items in a to represent its contextual feature vector, i.e.,

$$x_a = \frac{1}{|a|}\sum_{a'\in a}x_{a'},\ \forall a\in A, \tag{14}$$

where a' is a single item in a, and $x_{a'}$ represents the feature vector of item a'.

4.1.4. Baselines

We have implemented some well-known bandit algorithms for comparison, covering both traditional and contextual bandit algorithms. They are briefly described below.

C^{2}UCB [22]. It is a contextual combinatorial bandit algorithm that dynamically identifies various items of interest to users. Each item is represented as a feature vector, and each user is represented as a preference vector. In each round, the bandit algorithm selects items according to the item selection strategy.

TPBandit [10]. It is also a contextual combinatorial bandit, which models personalized online recommendations for e-commence as a two-phase MAB problem. The bandit algorithm generates an item subset in the first phase. Then, it treats the item subset generated in the first phase as the arms of MAB, outputs a recommended item, and calculates rewards through fine-grained implicit feedback from the user. It is worth noting that this method clusters the items in the first phase.

N-LinUCB [6]. It is a contextual bandit algorithm that pulls only one arm at a time. To recommend a set of items for the target user each time, it repeats the LinUCB algorithm N times in each round. The reward obtained in each round is the sum of the rewards generated by the selected N arms.

DC^{3}MAB − SIN ($DC^3MAB-SINgle$). It is a variant of our proposed DC^3MAB, which allocates a single bandit instance of DC^3MAB across all users, thus making similar predictions for all users. It benefits more from the wealth of training instances.

TS [19]. It is one of the oldest and most popular heuristics, with each arm corresponding to one item. It assumes that the reward for each arm follows a beta distribution, and the arm with the highest sample value will be pulled. In a typical recommendation task, the beta distribution involves two parameters, α' and β', set as the number of successes and failures, respectively. In our implementations, we focus on the relationship between multiple behavior types and user preferences and map the interaction behaviors to strong and weak preferences. Accordingly, we set the two parameters to strong preference-related and weak preference-related rewards.

UCB [3]. It is widely studied and applied, resulting in many variants. Following the principle of optimism in the face of uncertainty, it always chooses the arm with the highest upper confidence bound.

DynUCB [8]. It is a variant of UCB, which divides the user set into multiple clusters and customizes a bandit for each user cluster. However, it only selects one arm per round.

Table 3
The settings of several key parameters.

Datasets	α	k	d	#Super arms
IJCAI-15	0.1	48	8	14
Retailrocket	0.1	48	16	8
Yoochoose	0.1	48	25	6

4.2. Algorithm performance (RQ1)

The number of user clusters in the experiment is uniformly set to 48. α is the parameter for the importance of exploration. The experimental parameter settings are shown in Table 3. α' and β' are initialized to 1 in the TS algorithm. To verify the effectiveness and performance of the proposed algorithm, we compare it with different baselines in the following two aspects. It is worth noting that the F1 values in all tables are not calculated by inputting Precision and Recall values in the tables into Eq. (11), but the average values are calculated from the results of multiple rounds of experiments. There will be some minor differences between the two.

First, we compare our proposed algorithm with the contextual combinatorial bandit algorithms. Each round, the algorithms generate a recommendation list of length $N = 10$. The performance of different algorithms on the three datasets is shown in Table 4, from which we get the following observations.

(a) DC^3MAB performs best on these datasets. Compared with TPBandit, its F1 increased by 12.6%, 153.2% and 61.6%, respectively, with an average of 75.8%, while its CR increased by 5.4%, 100.9% and 56.5%, with an average of 54.3%. The variant $DC^3MAB - SIN$ performs second, validating the superiority of our proposed algorithm. In most cases, creating a separate bandit for each cluster performs better than creating a uniform bandit for all users.
(b) LinUCB recommends only one item per round. It needs to pull the arms several times to generate multiple recommended items, and this tedious process weakens the recommendation performance. In addition, N-LinUCB does not cluster similar users or similar items, and thus interaction information cannot be shared between users and items. Due to these factors, in most cases, its recommendation performance is not as good as C^2UCB with a user clustering strategy and TPBandit with an item partitioning strategy.
(c) TPBandit uses only the item partitioning strategy, and C^2UCB uses only the user clustering strategy, both of which do not perform as well as DC^3MAB with both strategies.

Second, we compare our proposed algorithm with the traditional bandits. When N is 1, the combinatorial bandits become traditional bandits, which provide each user with an individual item each round. Table 5 shows the performance of different non-combinatorial bandits on three datasets, from which we can get the following observations.

(a) When only one recommended item is generated per round, DC^3MAB still performs best compared to the baseline algorithms, while UCB, TS, and LinUCB perform poorly and receive few rewards. The main reason is that these algorithms model individual items as arms, resulting in poor item exploration.
(b) TPBandit and our algorithm DC^3MAB show substantial advantages over the UCB, TS and DynUCB algorithms, significantly improving recommendation performance. We can conclude that modeling the arm with a cluster of items performs better than modeling the arm with a single item.
(c) Table 4 and Table 5 show that the length of the recommendation list has little effect on the precision. It can be explained by the fact that there is some uncertainty in the bandit algorithms, i.e., the bandit algorithms may generate different arms even in the same context. For all bandit algorithms, the cumulative reward increases with the length of the recommendation list. Therefore, a diverse recommendation service can better improve user satisfaction.

4.3. Ablation study (RQ2)

(1) Impact of item partitioning

In this experiment, we will discuss the impact brought by item partitioning. We remove the item partitioning module from the DC^3MAB algorithm, i.e., each arm corresponds to one item. The new variant is called DC^3MAB w/o IP for comparison purposes. We select 1 000 users from IJCAI-15, Retailrocket and Yoochoose, respectively, with the same parameter settings as in Table 3. The results are showed in Table 6.

We can easily observe the advantages of our proposed item partitioning approach, especially on the first two datasets. Due to a large number of items, it is time-consuming for online recommendations to calculate the expected reward for each item per round, and it is also impractical to pull each arm to obtain the reward distribution for each arm. Therefore, modeling an item as an arm is inappropriate in practical recommendation scenarios. Meanwhile, the collaborative filtering-based item partitioning approach is simple and effective. It helps to quickly select a subset of items that may be of interest to the target user based on the current interaction behavior.

(2) Impact of multi-class reward mechanism

To verify the generality and effectiveness of our proposed reward mechanism, we implement two reward mechanisms, the traditional Bernoulli-based reward mechanism (abbreviated as 0–1) and our proposed multi-class reward mechanism (abbreviated as MC). We use these two mechanisms in three bandit algorithms, including DC^3MAB, N-LinUCB, and TPBandit. We set the same parameter values in Table 3. The experimental results of these two reward mechanisms in different algorithms are shown in Table 7, from which we obtain the following observations.

Our proposed MC reward mechanism can be flexibly applied in various bandit algorithms. In these three bandit algorithms, the MC reward mechanism obtains higher precision and F1 than the 0–1 reward mechanism. Moreover, in most cases, the MC reward mechanism obtains a higher recall.

The MC reward mechanism affects different bandit algorithms, especially those that model individual items as arms. The main reason is that the MC reward mechanism provides more information about the user's preference for the bandit algorithms. For example, in the Yoochoose dataset, the MC reward mechanism helps N-LinUCB, TPBandit, and DC^3MAB algorithms improve the F1 metric by 133%, 62%, and 23%, respectively.

The MC reward mechanism performs differently on different datasets, especially for the datasets with rich interaction information such as IJCAI. For example, for the DC^3MAB algorithm, its F1 improvements on the IJCAI, Retailrocket, and Yoochoose datasets are 51%, 14%, and 23%, respectively. The N-LinUCB's F1 improvements on the IJCAI, Retailrocket and Yoochoose datasets are 267%, 375%, and 133%, respectively. The TPBandit's F1 improvements on the IJCAI, Retailrocket and Yoochoose datasets are 147%, 54%, and 62%, respectively.

Table 4
The performance comparison of DC^3MAB with other contextual combinatorial bandits ($N = 10$, best bold, second best in blue).

Datasets	Metrics	C^2UCB	N-LinUCB	TPBandit	DC^3MAB	$DC^3MAB - SIN$
IJCAI-15	Precision	0.0009	0.0030	0.0837	**0.1427**	0.1326
	Recall	0.0003	0.0010	**0.0458**	0.0435	0.0426
	F1	0.0004	0.0010	0.0524	**0.0590**	0.0567
	CR	282	254	7302	**7697**	7427
Retailrocket	Precision	0.0011	0.0030	0.0265	**0.0734**	0.0589
	Recall	0.0006	0.0021	0.0253	**0.0591**	0.0511
	F1	0.0007	0.0019	0.0222	**0.0562**	0.0466
	CR	184	171	2810	**5644**	4732
Yoochoose	Precision	0.0030	0.0099	0.0313	**0.0521**	0.0499
	Recall	0.0008	0.0039	0.0406	**0.0645**	0.0620
	F1	0.0013	0.0042	0.0333	0.0538	**0.0620**
	CR	536	852	8102	**12679**	12031

Table 5
The performance comparison of DC^3MAB with other traditional bandits ($N = 1$).

Datasets	Metrics	UCB	TS	TPBandit	DynUCB	LinUCB	DC^3MAB
IJCAI-15	Precision	0.0011	0.0007	0.0825	0.0026	0.0010	**0.1485**
	Recall	0.0001	0.000009	0.0090	0.0003	0.0001	**0.0166**
	F1	0.0002	0.0002	0.0160	0.0005	0.0002	**0.0295**
	CR	28	18	1308	77	56	**3019**
Retailrocket	Precision	0.0012	0.0013	0.0192	0.0088	0.0010	**0.0772**
	Recall	0.0002	0.0002	0.0043	0.0019	0.0002	**0.0168**
	F1	0.0004	0.0003	0.0069	0.0030	0.0004	**0.0267**
	CR	12	11	413	198	50	**1478**
Yoochoose	Precision	0.0019	0.0014	0.0185	0.0356	0.0046	**0.0500**
	Recall	0.0003	0.0002	0.0027	0.0062	0.0006	**0.0760**
	F1	0.0004	0.0003	0.0047	0.0100	0.0010	**0.0130**
	CR	59	40	560	1069	137	**1445**

Table 6
The effect of item partitioning ($N = 10$).

Datasets	Algorithm	Precision	Recall	F1	CR
IJCAI-15	DC^3MAB	**0.1427**	**0.0435**	**0.0590**	**7697**
	DC^3MAB w/o IP	0.0010	0.0011	0.0010	192
Retailrocket	DC^3MAB	**0.0734**	**0.0591**	**0.0562**	**5644**
	DC^3MAB w/o IP	0.0031	0.007	0.0041	643
Yoochoose	DC^3MAB	**0.0521**	**0.0645**	**0.0538**	**12679**
	DC^3MAB w/o IP	0.0163	0.0257	0.0188	4947

Table 7
The effect of multi-class reward mechanism.

Algorithm	Dataset	Mechanism	Precision	Recall	F1
DC^3MAB	IJCAI-15	MC	**0.1427**	0.0435	**0.0590**
		0–1	0.0522	**0.0451**	0.0391
	Retailrocket	MC	**0.0734**	**0.0591**	**0.0562**
		0–1	0.0385	0.0118	0.0493
	Yoochoose	MC	**0.0521**	0.0645	**0.0538**
		0–1	0.0345	**0.0726**	**0**.0439
N-LinUCB	IJCAI-15	MC	**0.0030**	**0.0010**	**0.0010**
		0–1	0.0002	**0.0010**	0.0003
	Retailrocket	MC	**0.0030**	**0.0021**	**0.0019**
		0–1	0.0002	0.0010	0.0004
	Yoochoose	MC	**0.0099**	**0.0039**	**0.0042**
		0–1	0.0013	0.0036	0.0018
TPBandit	IJCAI-15	MC	**0.0837**	**0.0458**	**0.0524**
		0–1	0.0278	0.0343	0.0244
	Retailrocket	MC	**0.0265**	0.0253	**0.0222**
		0–1	0.0115	**0.0319**	0.0144
	Yoochoose	MC	**0.0313**	**0.0406**	**0.0333**
		0–1	0.0157	0.0351	0.0206

4.4. Effect of important parameters (RQ3)

This section mainly analyses the impact of essential parameters in the DC^3MAB algorithm on the recommendation performance, including the dimension of the contextual feature vector d, the number of user clusters k, and the length of the behavior sequence L.

4.4.1. Dimension of contextual feature vector

The contextual feature dimension d affects the recommendation performance. Its length determines the degree of differentiation of user preferences. We divide the users into 48 clusters. The effects of different dimensions of contextual feature on the three datasets are shown in Fig. 3, Fig. 4, and Fig. 5 respectively.

On the IJCAI-15 dataset, as the dimension of the feature vector increases, the four evaluation metrics of precision, recall, F1, and CR show the same change trend. They all fall first, then rise, and reach a minimum at $d = 6$, which means that the algorithm performs worst at $d = 6$.

On the Retailrocket dataset, the value of CR increases steadily as the dimension of the feature vector increases. The recall and precision metrics fluctuate with increasing dimensionality. The trends of F1 and precision are consistent. Meanwhile, the value of recall increases linearly when $d > 8$.

On the Yoochoose dataset, these four metrics show the same trend. Moreover, on this dataset, the values of these evaluation metrics are very small, and the recall metric is greater than the precision metric.

In summary, for the three datasets, the four evaluation metrics show different changing patterns with the change of the dimension of the contextual feature vector. However, it is inevitable that the feature dimension greatly influences the recommendation effect, and the optimal feature vector dimension is related to the characteristics of the datasets. For the IJCAI-15 dataset, the scale of items is large, and the interaction matrix is sparse, so user features are challenging to extract. Therefore, a smaller dimension brings better results.

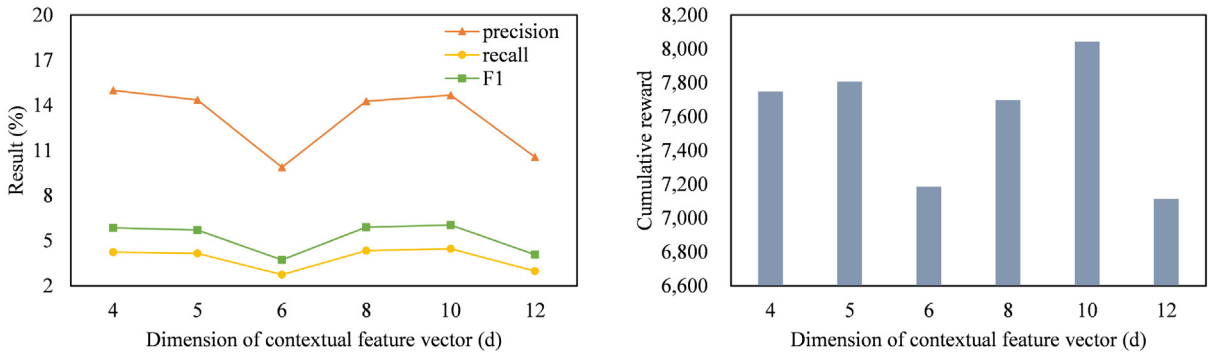

Fig. 3. The impact of contextual feature dimension on the IJCAI-15 dataset ($k = 48$).

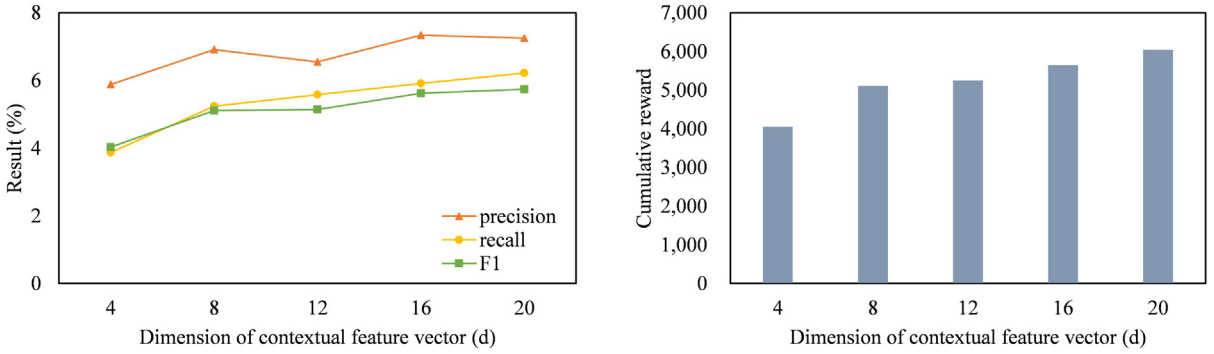

Fig. 4. The impact of contextual feature dimension on the Retailrocket dataset ($k = 48$).

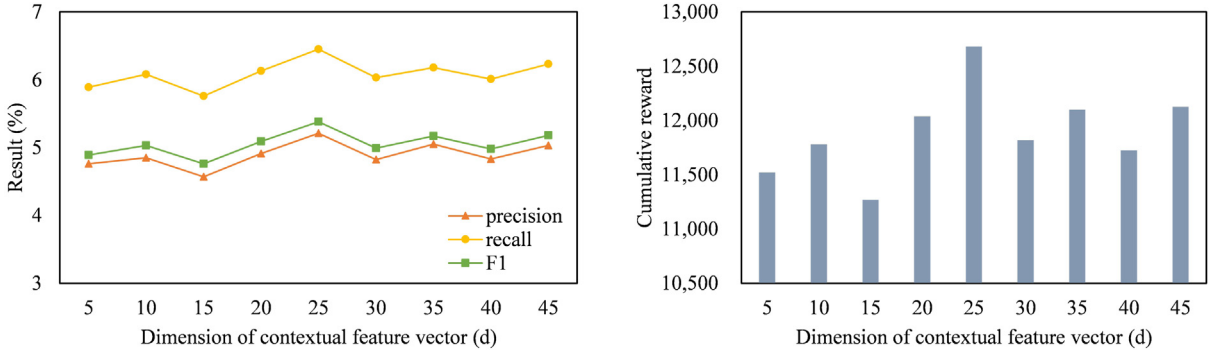

Fig. 5. The impact of contextual feature dimension on the Yoochoose dataset ($k = 48$).

4.4.2. Number of user clusters

Different numbers of user clusters k will affect the recommendation performance. We fix the contextual feature dimension and set different k values to observe its impact on recommendation performance. Other parameter settings are shown in Table 3. Furthermore, we consider the special case of $k = 1$, the same as the $DC^3MAB - SIN$ algorithm. The results are shown in Fig. 6, Fig. 7, and Fig. 8.

For the IJCAI-15 dataset, the value of k has little effect on the four metrics. The algorithm obtains the maximum cumulative reward when $k = 24$. We infer that the preferences of the test users on this dataset are relatively similar since recommendation performance does not change significantly regardless of the value of k.

For the Retailrocket dataset, precision and F1 show similar trends under different k values. When $k = 36$, the four metric values all reach the maximum value; among four metrics, the fluctuation of recall is more pronounced with the change of the k value.

For the Yoochoose dataset, the larger the number of user clusters, the better the recommendation performance. However, when the number of user clusters is greater than 24, the obtained recommendation results remain stable, which also shows the effectiveness of user clustering.

Generally, CR is more sensitive to k, as it varies widely with the value of k in the figures corresponding to these three datasets. The k influences different data sets' precision, recall, and F1. For example, for the IJCAI-15 dataset, the k has little effect on them, while for the other two datasets, the three metrics have similar changing trends when k increases. Therefore, we can conclude that user clustering does not significantly improve recommendation performance for datasets with high similarity between

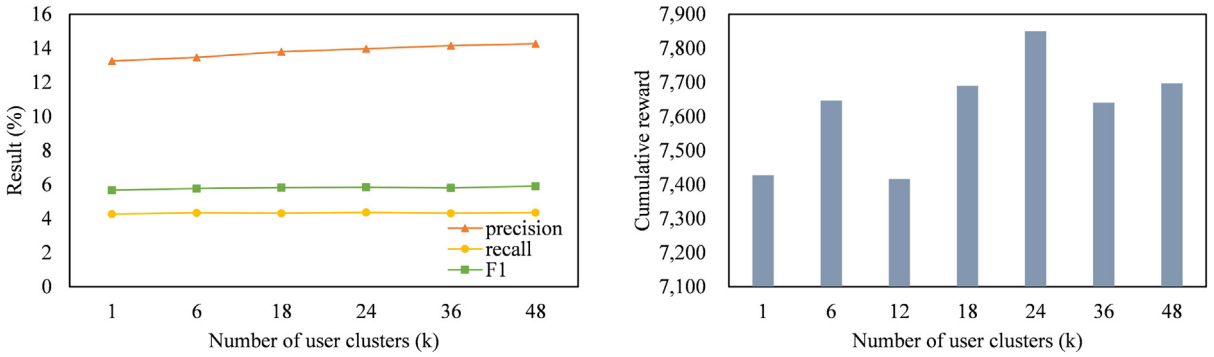

Fig. 6. The impact of the number of user clusters on the IJCAI-15 dataset ($d = 10$).

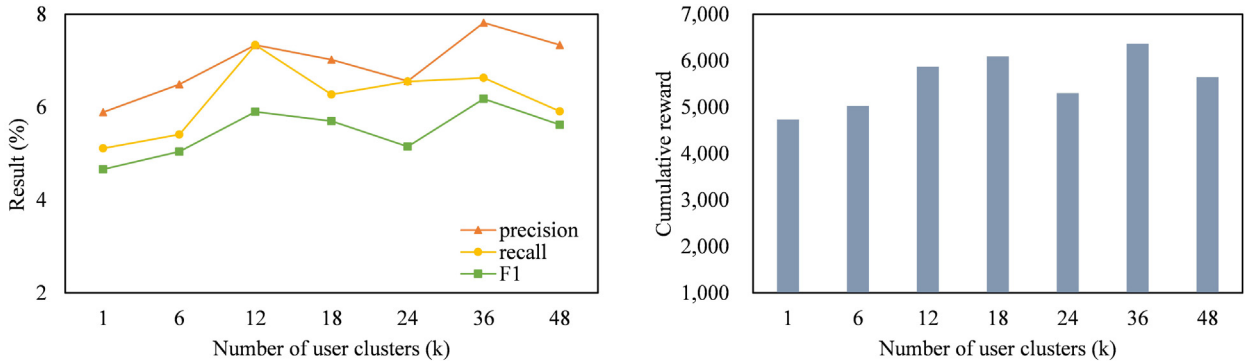

Fig. 7. The impact of the number of user clusters on the Retailrocket dataset ($d = 16$).

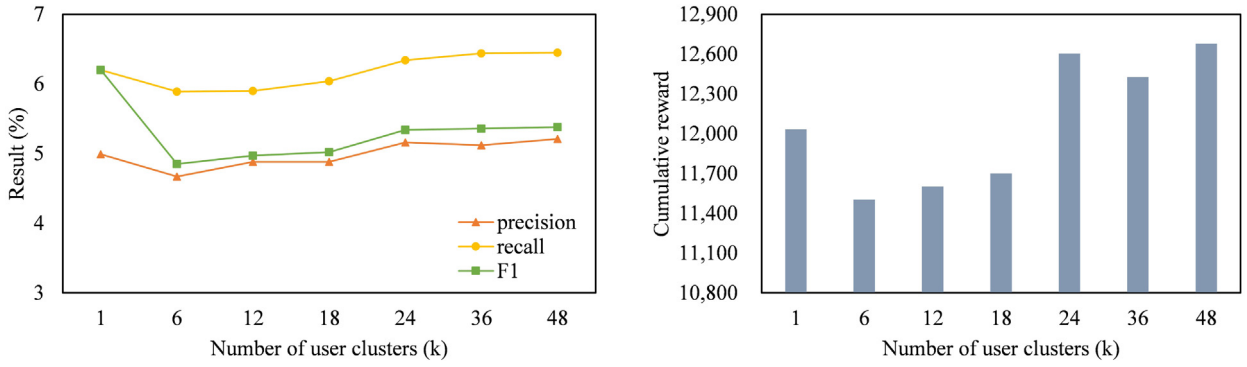

Fig. 8. The impact of the number of user clusters on the Yoochoose dataset ($d = 25$).

users, and it even increases the computational complexity. However, when interaction data is sparse, clustering users is beneficial to improve recommendation accuracy through collaboration between similar users.

4.4.3. Length of user behavior sequence

The user behavior sequence length L has a significant impact on recommendation performance. In this experiment, we set the dimension of the contextual feature vector to 10 for the IJCAI-15 dataset, 16 for the Retailrocket dataset, and 25 for the Yoochoose dataset. We initially divide the test users into 24 clusters. The sequence length L is set to $[len, len{+}interval]$. For the IJCAI-15 and Yoochoose datasets, we set the interval to 10. For Retailrocket, due to the specificity of its data distribution, we choose different sequence lengths for comparison. The results are shown in Table 8.

For the IJCAI-15 dataset, the recommendation precision increases as the sequence length when the sequence length is [10,40]. However, it starts to decrease when $L = 40$. The recommendation precision for the Retailrocket and Yoochoose datasets does not improve significantly with the increase in sequence length.

Theoretically, as the number of arm pulls increases, the bandit algorithm can better infer the reward probability for each arm, i.e., as the user's behavior records increase, the algorithm is more familiar with the user's preference, thereby improving the recommendation performance. However, the experimental results show an opposite phenomenon. As the sequence length increases, the recall decreases significantly. There are two possible reasons.

First, some behavior sequences contain multiple sessions with long time intervals between sessions. Therefore, although the number of behavior records increases, it does not bring helpful information but more irrelevant information for the recommendation.

Second, the increase in hits is lower than the increase in sequence length, which ultimately leads to this situation. In Eq. (9),

Table 8
The impact of the length of user behavior sequence ($k = 24$, and $d = 10$, 16, 25, respectively).

Dataset	L	Precision	Recall	F1	CR
IJCAI-15	[10,20)	0.0981	**0.0268**	**0.0373**	6287
	[20,30)	**0.1190**	0.0255	**0.0373**	13725
	[30,40)	0.1175	0.0192	0.0301	20986
	[40,50)	0.1082	0.0137	0.0224	**25126**
Retailrocket	[10,20)	**0.0656**	**0.0655**	**0.0515**	5300
	[20,50)	0.0643	0.038	0.0416	**14778**
Yoochoose	[10,20)	**0.0516**	**0.0634**	**0.0534**	12604
	[20,30)	0.0506	0.0419	0.0441	22563
	[30,40)	0.0479	0.0279	0.0338	29403
	[40,50)	0.0496	0.0231	0.0302	**38071**

$\left|\bar{I}_{ut} \cap \bar{S}_u\right|$ represents the hits at each time, and $\left|\bar{S}_u\right|$ is positively correlated with the length of the user behavior sequence. The increase rate of the hit number is lower than the increase rate of the sequence length, resulting in a decreasing recall trend. Moreover, we find that the hit number increases with the increase of the sequence length, which also can be verified by the growth rate of the cumulative reward.

5. Conclusion

Modeling the recommendation task as a bandit problem is an effective way to cope with the EE problem in recommender systems. However, it is often constrained by the traditional bandit setting. In this paper, We propose an algorithm called DC^3MAB that solves several critical problems due to the large scale of items and users and their sparse interactions. We design a dynamic user clustering strategy to achieve cooperation of similar users, enabling the capture of user preferences even in the case of sparse interactions. We define the concept of super arm and propose to model item subsets as arms by a dynamic item partitioning approach, which improves the diversity of recommendations and avoids the computational complexity associated with large-scale items. In addition, we put forward a multi-class reward mechanism based on fine-grained implicit feedback to replace the Bernoulli-based reward mechanism used in the traditional bandits, which also helps better capture user preferences. Most importantly, these three key components can be flexibly and independently applied in other MAB-based recommender systems.

Experimental results on three public datasets demonstrate that our proposed algorithm DC^3MAB obtain better performance than the baselines in terms of precision, recall, F1 score, and cumulative reward. In future work, we will explore the regret analysis of DC^3MAB to verify its theoretical feasibility. We will also consider its extension and investigate clustering strategies regarding users, items, and bandits.

CRediT authorship contribution statement

Cairong Yan: Conceptualization, Methodology, Validation, Writing – review & editing. **Haixia Han:** Software, Formal analysis, Investigation, Data curation, Writing – original draft. **Yanting Zhang:** Visualization, Funding acquisition. **Dandan Zhu:** Funding acquisition, Writing – review & editing. **Yongquan Wan:** Validation, Writing – review & editing.

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 work is partly supported by the National Natural Science Foundation of China (62001289) and the Shanghai Sailing Program, China (21YF1401300).

References

[1] C. Li, Q. Wu, H. Wang, When and whom to collaborate with in a changing environment: A collaborative dynamic bandit solution, in: Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR, 2021, pp. 1410–1419.
[2] A. Barraza-Urbina, The exploration-exploitation trade-off in interactive recommender systems (RecSys), in: Proceedings of the 11th ACM Conference on Recommender Systems, RecSys, 2017, pp. 431–435.
[3] P. Auer, N. Cesa-Bianchi, P. Fischer, Finite-time analysis of the multiarmed bandit problem, Mach. Learn. 47 (2) (2002) 235–256.
[4] Q. Wang, C. Zeng, W. Zhou, T. Li, S. Iyengar, L. Shwartz, G.Y. Grabarnik, Online interactive collaborative filtering using multi-armed bandit with dependent arms, IEEE Trans. Knowl. Data Eng. 31 (8) (2019) 1569–1580.
[5] J. Zhang, X. Luo, Y. Zhou, W. Ruan, Y. Jiang, Z. Hao, Two-way negotiation for intelligent hotel reservation based on multiagent: The model and system, Knowl.-Based Syst. 161 (2018) 78–89.
[6] 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, WWW, 2010, pp. 661–670.
[7] A. Lacerda, Multi-objective ranked bandits for recommender systems, Neurocomputing 246 (2017) 12–24.
[8] T.T. Nguyen, H.W. Lauw, Dynamic clustering of contextual multi-armed bandits, in: Proceedings of the 23rd ACM International Conference on Information and Knowledge Management, CIKM, 2014, pp. 1959–1962.
[9] K. Christakopoulou, A. Banerjee, Learning to interact with users: A collaborative-bandit approach, in: Proceedings of the 2018 SIAM International Conference on Data Mining, ICDM, 2018, pp. 612–620.
[10] C. Yan, H. Han, Z. Wang, Y. Zhang, Two-phase multi-armed bandit for online recommendation, in: Proceedings of the 8th International Conference on Data Science and Advanced Analytics, DSAA, 2021, pp. 1–8.
[11] C. Yan, J. Xian, Y. Wan, P. Wang, Modeling implicit feedback based on bandit learning for recommendation, Neurocomputing 447 (2021) 244–256.
[12] M. Gan, O.-C. Kwon, A knowledge-enhanced contextual bandit approach for personalized recommendation in dynamic domains, Knowl.-Based Syst. 251 (2022) 109158.
[13] X. Xu, H. Xie, J.C. Lui, Generalized contextual bandits with latent features: Algorithms and applications, IEEE Trans. Neural Netw. Learn. Syst. (2021) http://dx.doi.org/10.1109/TNNLS.2021.3124603.
[14] A. Said, S. Berkovsky, E.W. De Luca, J. Hermanns, Challenge on context-aware movie recommendation: CAMRa2011, in: Proceedings of the 5th ACM Conference on Recommender Systems, RecSys, 2011, pp. 385–386.
[15] X. Zhang, H. Xie, H. Li, J. C.S. Lui, Conversational contextual bandit: Algorithm and application, in: Proceedings of the World Wide Web Conference, WWW, 2020, pp. 662–672.
[16] X. Xu, F. Dong, Y. Li, S. He, X. Li, Contextual-bandit based personalized recommendation with time-varying user interests, in: Proceedings of the AAAI Conference on Artificial Intelligence, AAAI, 2020, pp. 6518–6525.
[17] S. Agrawal, N. Goyal, Thompson sampling for contextual bandits with linear payoffs, in: Proceedings of the International Conference on Machine Learning, ICML, 2013, pp. 127–135.
[18] C.-C. Hsieh, J. Neufeld, T. King, J. Cho, Efficient approximate thompson sampling for search query recommendation, in: Proceedings of the 30th Annual ACM Symposium on Applied Computing, SAC, 2015, pp. 740–746.
[19] O. Chapelle, L. Li, An empirical evaluation of thompson sampling, in: Proceedings of the Advances in Neural Information Processing Systems, NIPS, Vol. 24, 2011, pp. 1–9.
[20] D.K. Mahajan, R. Rastogi, C. Tiwari, A. Mitra, Logucb: an explore-exploit algorithm for comments recommendation, in: Proceedings of the 21st ACM International Conference on Information and Knowledge Management, CIKM, 2012, pp. 6–15.
[21] D. Bouneffouf, A. Bouzeghoub, A.L. Gançarski, A contextual-bandit algorithm for mobile context-aware recommender system, in: Proceedings

of the International Conference on Neural Information Processing, ICONIP, 2012, pp. 324–331.
[22] L. Qin, S. Chen, X. Zhu, Contextual combinatorial bandit and its application on diversified online recommendation, in: Proceedings of the 2014 SIAM International Conference on Data Mining, ICDM, 2014, pp. 461–469.
[23] S. Li, A. Karatzoglou, C. Gentile, Collaborative filtering bandits, in: Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR, 2016, pp. 539–548.
[24] N. Craswell, O. Zoeter, M. Taylor, B. Ramsey, An experimental comparison of click position-bias models, in: Proceedings of the 2008 International Conference on Web Search and Data Mining, WSDM, 2008, pp. 87–94.
[25] B. Kveton, C. Szepesvari, Z. Wen, A. Ashkan, Cascading bandits: Learning to rank in the cascade model, in: Proceedings of the International Conference on Machine Learning, ICML, 2015, pp. 767–776.
[26] S. Zong, H. Ni, K. Sung, N.R. Ke, Z. Wen, B. Kveton, Cascading bandits for large-scale recommendation problems, in: Proceedings of 32nd Conference on Uncertainty in Artificial Intelligence, UAI, 2016, pp. 835–844.
[27] J. Komiyama, J. Honda, H. Nakagawa, Optimal regret analysis of thompson sampling in stochastic multi-armed bandit problem with multiple plays, in: Proceedings of the International Conference on Machine Learning, ICML, 2015, pp. 1152–1161.
[28] W. Chen, Y. Wang, Y. Yuan, Combinatorial multi-armed bandit: General framework and applications, in: Proceedings of the International Conference on Machine Learning, ICML, 2013, pp. 151–159.
[29] B. Kveton, Z. Wen, A. Ashkan, C. Szepesvari, Combinatorial cascading bandits, in: Proceedings on Advances in Neural Information Processing Systems, NIPS, 2015, pp. 1450–1458.
[30] C. Gentile, S. Li, G. Zappella, Online clustering of bandits, in: Proceedings of International Conference on Machine Learning, ICML, 2014, pp. 757–765.
[31] S. Li, S. Zhang, Online clustering of contextual cascading bandits, in: Proceedings of the AAAI Conference on Artificial Intelligence, AAAI, 2018, pp. 3554–3561.
[32] L. Yang, B. Liu, L. Lin, F. Xia, K. Chen, Q. Yang, Exploring clustering of bandits for online recommendation system, in: Proceedings of the 14th ACM Conference on Recommender Systems, RecSys, 2020, pp. 120–129.
[33] J. Sanz-Cruzado, P. Castells, E. López, A simple multi-armed nearest-neighbor bandit for interactive recommendation, in: Proceedings of the 13th ACM Conference on Recommender Systems, RecSys, 2019, pp. 358–362.
[34] T. Joachims, L. Granka, B. Pan, H. Hembrooke, G. Gay, Accurately interpreting clickthrough data as implicit feedback, in: Proceedings of the 28th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR, 2017, pp. 154–161.
[35] M. Zoghi, T. Tunys, M. Ghavamzadeh, B. Kveton, C. Szepesvari, Z. Wen, Online learning to rank in stochastic click models, in: Proceedings of the International Conference on Machine Learning, ICML, 2017, pp. 4199–4208.
[36] I. Schoinas, C. Tjortjis, Musif: A product recommendation system based on multi-source implicit feedback, in: Proceedings of the IFIP International Conference on Artificial Intelligence Applications and Innovations, AIAI, 2019, pp. 660–672.
[37] C. Gao, X. He, D. Gan, X. Chen, F. Feng, Y. Li, T.-S. Chua, D. Jin, Neural multi-task recommendation from multi-behavior data, in: Proceedings of the 35th International Conference on Data Engineering, ICDE, 2019, pp. 1554–1557.
[38] X. He, H. Zhang, M.-Y. Kan, T.-S. Chua, Fast matrix factorization for online recommendation with implicit feedback, in: Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR, 2016, pp. 549–558.