Estimation–Action–Reflection: Towards Deep Interaction Between Conversational and Recommender Systems

Wenqiang Lei[1], Xiangnan He[2*], Yisong Miao[1], Qingyun Wu[3], Richang Hong[4], Min-Yen Kan[1], Tat-Seng Chua[1]
[1]National University of Singapore, [2]University of Science and Technology of China
[3]University of Virginia, [4]Hefei University of Technology
wenqianglei@gmail.com,xiangnanhe@gmail.com,miaoyisong@gmail.com,qw2ky@virginia.edu
hongrc@hfut.edu.cn,kanmy@comp.nus.edu.sg,chuats@comp.nus.edu.sg

ABSTRACT

Recommender systems are embracing conversational technologies to obtain user preferences dynamically, and to overcome inherent limitations of their static models. A successful *Conversational Recommender System* (CRS) requires proper handling of interactions between conversation and recommendation. We argue that three fundamental problems need to be solved: 1) what questions to ask regarding item attributes, 2) when to recommend items, and 3) how to adapt to the users' online feedback. To the best of our knowledge, there lacks a unified framework that addresses these problems.

In this work, we fill this missing interaction framework gap by proposing a new CRS framework named *Estimation–Action–Reflection*, or EAR, which consists of three stages to better converse with users. (1) Estimation, which builds predictive models to estimate user preference on both items and item attributes; (2) Action, which learns a dialogue policy to determine whether to ask attributes or recommend items, based on Estimation stage and conversation history; and (3) Reflection, which updates the recommender model when a user rejects the recommendations made by the Action stage. We present two conversation scenarios on binary and enumerated questions, and conduct extensive experiments on two datasets from Yelp and LastFM, for each scenario, respectively. Our experiments demonstrate significant improvements over the state-of-the-art method CRM [32], corresponding to fewer conversation turns and a higher level of recommendation hits.

CCS CONCEPTS

• **Information systems → Users and interactive retrieval**; **Recommender systems**; *Personalization*; • **Human-centered computing** → *Interactive systems and tools.*

KEYWORDS

Conversational Recommendation; Interactive Recommendation; Recommender System; Dialogue System

*Xiangnan He is the Corresponding Author.

WSDM '20, February 3–7, 2020, Houston, TX, USA

ACM ISBN 978-1-4503-6822-3/20/02…$15.00
https://doi.org/10.1145/3336191.3371769

ACM Reference Format:
Wenqiang Lei, Xiangnan He, Yisong Miao, Qingyun Wu, Richang Hong, Min-Yen Kan, Tat-Seng Chua. 2020. Estimation–Action–Reflection: Towards Deep Interaction Between Conversational and Recommender Systems. In *The Thirteenth ACM International Conference on Web Search and Data Mining (WSDM'20), February 3–7, 2020, Houston, TX, USA.* ACM, NY, NY, USA, 9 pages. https://doi.org/10.1145/3336191.3371769

1 INTRODUCTION

Recommender systems are emerging as an important means of facilitating users' information seeking [6, 17, 20, 30]. However, much of such prior work in the area solely leverages the offline historical data to build the recommender model (henceforth, the *static recommender system*). This offline focus causes the recommender to suffer from an inherent limitation in the optimization of offline performance, which may not necessarily match online user behavior. User preference can be diverse and often drift with time; and as such, it is difficult to know the exact intent of a user when he uses a service even when the training data is sufficient.

The rapid development of conversational techniques [19, 22, 23, 26, 35] brings an unprecedented opportunity that allows a recommender system to dynamically obtain user preferences through conversations with users. This possibility is envisioned as the *conversational recommender system* (CRS), for which the community has started to expend effort in exploring its various settings. [40] built a conversational search engine by focusing on document representation. [23] developed a dialogue system to suggest movies for cold start users, contributing to language understanding and generation for the purpose of recommendation, but does not consider modeling users' interaction histories (e.g., clicks, ratings). In contrast, [9] does considers user click history in recommending, but their CRS only handles single-round recommendation. That is, their model considers a scenario in which the CRS session terminates after making a single recommendation, regardless of whether the recommendation is satisfactory or not. While a significant advance, we feel this scenario is unrealistic in actual deployments.

In particular, we believe CRS models should inherently adopt a *multi-round* setting: a CRS converses with a user to recommend items based on his click history (if any). At each round, the CRS is allowed to choose two types of actions — either explicitly asking whether a user likes a certain item attribute or recommending a list of items. In a session, the CRS may alternate between these actions multiple times, with the goal of finding desirable items while minimizing the number of interactions. This multi-round

setting is more challenging than the single-round setting, as the CRS needs to strategically plan its actions. The key in performing such planning, from our perspective, lies in the interaction between the conversational component (CC; responsible for interacting with the user) and the recommender component (RC; responsible for estimating user preference – e.g., generating the recommendation list). We summarize three fundamental problems toward the deep interaction between CC and RC as follows:

- *What attributes to ask?* A CRS needs to choose which attribute to ask the user about. For example, in music recommendation, it may ask "Would you like to listen to classical music?", expecting a binary yes/no response[1]. If the answer is "yes", it can focus on items containing the attribute, benefiting the RC by reducing uncertainty in item ranking. However, if the answer is "no", the CRS expends a conversation turn with less gain to the RC. To achieve the goal of hitting the right items in fewer turns, the CC must consider whether the user will like the asked attribute. This is exactly the job of the RC which scrutinizes the user's historical behavior.
- *When to recommend items?* With sufficient certainty, the CC should push the recommendations generated by the RC. A good timing to push recommendations should be when 1) the candidate space is small enough; when 2) asking additional questions is determined to be less useful or helpful, from the perspective of either information gain or user patience; and when 3) the RC is confident that the top recommendations will be accepted by the user. Determining the appropriate timing should take both the conversation history of the CC and the preference estimation of the RC into account.
- *How to adapt to users' online feedback?* After each turn, the user gives feedback; i.e., "yes"/"no" to a queried attribute, or an "accept"/"reject" the recommended items. (1) For "yes" on the attribute, both user profile and item candidates need to be updated to generate better recommendations; this requires the offline RC training to take such updates into account. (2) For "no', the CC needs to adjust its strategy accordingly. (3) If the recommended items are rejected, the RC model needs to be updated to incorporate such a negative signal. Although adjustments seem only to impact either the RC or the CC, we show that such actions impact both.

Towards the deep interaction between CC and RC, we propose a new solution named *Estimation–Action–Reflection* (EAR), which consists of three stages. Note that the stages do not necessarily align with each of the above problems. (a) Estimation, which builds predictive models offline to estimate user preference on items and item attributes. Specifically, we train a factorization machine [29] (FM) using user profiles and item attributes as input features. Our Estimation stage builds in two novel advances: 1) the joint optimization of FM on the two tasks of item prediction and attribute prediction, and 2) the adaptive training of conversation data with online user feedback on attributes. (b) Action, which learns the conversational strategy that determines whether to ask or recommend, and what attribute to ask. We train a policy network with reinforcement learning, optimizing the reward of shorter turns and successful recommendations based on the FM's estimation of user preferred items and attributes, and the dialogue history. (c) Reflection, which adapts the CRS with user's online feedback. Specifically, when a user rejects the recommended items, we construct new training triplets by treating the items as negative instances and update the FM in an online manner. In summary, the main contributions of this work are as follows:

- We comprehensively consider a multi-round CRS scenario that is more realistic than previous work, highlighting the importance of researching into the interactions between the RC and CC to build an effective CRS.
- We propose a three-stage solution, EAR, integrating and revising several RC and CC techniques to construct a solution that works well for the conversational recommendation.
- We build two CRS datasets by simulating user conversations to make the task suitable for offline academic research. We show our method outperforms several state-of-the-art CRS methods and provide insight on the task.

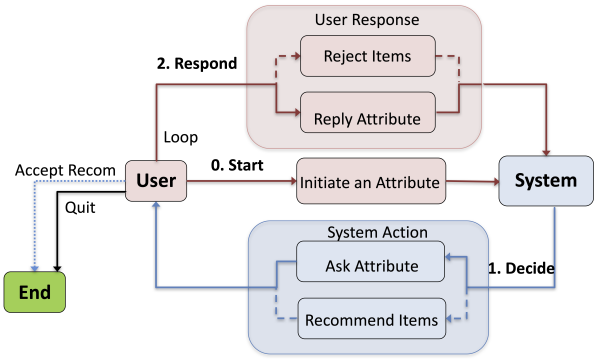

Figure 1: The workflow of our multi-round conversational recommendation scenario. The system may recommend items multiple times, and the conversation ends only if the user accepts the recommendation or chooses to quit.

2 MULTI-ROUND CONVERSATIONAL RECOMMENDATION SCENARIO

Following [9], we denote one trial of recommendation as a *round*. This paper considers conversational recommendation as an inherently *multi-round* scenario, where a CRS interacts with the user by asking attributes and recommending items multiple times until the task succeeds or the user leaves. To distinguish the two, we term the setting *single-round* where the CRS only makes recommendations once, ending the session regardless of the outcome, as in [9, 32].

We now introduce the notation used to formalize our setting. Let $u \in \mathcal{U}$ denote a user u from the user set $\mathcal{U}$ and $v \in \mathcal{V}$ denote an item v from the item set $\mathcal{V}$. Each item v is associated with a set of attributes $\mathcal{P}_v$ which describe its properties, such as music genre "classical" or "jazz" for songs in LastFM, or tags such as "nightlife", "serving burgers", or "serving wines" for businesses in Yelp. We denote the set of all attributes as $\mathcal{P}$ and use p to denote a specific attribute. Following [32, 40], a CRS session is started with u's specification of a preferred attribute p^0, then the CRS filters out candidate items that contain the preferred attribute p^0. Then in

[1]Note that it is possible to compose questions eliciting an enumerated response; i.e., "Which music genre would you consider? I have pop, funk ...". However, this is a design choice depending on the domain requirements. In describing our method, we consider the basic single-attribute case. However in experiments, we also justify the effectiveness of EAR in asking such enumerated questions on Yelp. For the purpose of exposition, we have chosen to avoid open questions that do not constrain user response for now. Even interpreting user responses to such questions is considered a challenging task [5].

each turn t ($t = 1, 2, ..., T$; T denotes the last turn of the session), the CRS needs to choose an action: *recommend* or *ask*:

- If the ACTION is *recommend*, we denote the recommended item list $\mathcal{V}^t \subset \mathcal{V}$ and the action as a_{rec}. Then the user examines whether $\mathcal{V}^t$ contains his desired item. If the feedback is positive, this session succeeds and can be terminated. Otherwise, we mark $\mathcal{V}^t$ as *rejected* and move to the next round.
- If the ACTION is *ask* (where the asked attribute is denoted as $p^t \in \mathcal{P}$ and the action as $a_{ask}(p^t)$), the user states whether he prefers items that contain the attribute p^t or not. If the feedback is positive, we add p^t into $\mathcal{P}_u$ to denote the preferred attributes the user in the current session. Otherwise, we mark p^t as *rejected*; regardless of rejection or not, we move to the next turn.

This whole process naturally forms a interaction loop (Figure 1) where the CRS may ask zero to many questions before making recommendations. A session terminates if a user accepts the recommendations or leaves due to his impatience. We set the main goal of the CRS as making desired recommendations within as few rounds as possible.

3 PROPOSED METHODS

EAR consists of a recommendation and conversation component (RC and CC) which interact intensively in the three–stage conversational process. The system starts working at the *estimation* stage where the RC ranks candidate items and item attributes for the user, so as to support the action decision of the CC. After the *estimation* stage, the system moves to the *action* stage where the CC decides whether to choose an attribute to ask, or make a recommendation according to the ranked candidates and attributes, and the dialogue history. If the user likes the attribute asked by the RC, the CC feeds this attribute back to the RC to make a new *estimation* again; otherwise, the system stays at the *action* stage: updates the dialogue history and chooses another action. Once a recommendation is rejected by a user, the CC sends the rejected items back to RC, triggering the *reflection* stage where the RC adjusts its estimations. After that, the system enters the *estimation* stage again.

3.1 Estimation

As discussed before, the multi-round conversational scenario brings in new challenges to the traditional RC. Specifically, the CC interacts with a user u and accumulates evidence on his preferred attributes, denoted as $\mathcal{P}_u = \{p_1, p_2, .., p_n\}$[2]. Importantly, different from traditional recommendation methods [17, 30], the RC here needs to make full use of $\mathcal{P}_u$ aiming to accurately predict both user's the preferred items and preferred attributes. These two goals exert positive influence on EAR, where the first directly contributes to success rate of recommendation, and the second guides the CC to choose better attributes to ask users so as to shorten the conversation. In the following, we first introduce the basic form of the recommendation method, followed by detail on how we adapt our proposed method to achieve both goals simultaneously.

3.1.1 Basic Recommendation Method. we choose the factorization machine (FM) [29] as our predictive model due to its success and wide usage in recommendation tasks. However, FM considers all pairwise interactions between input features, which is costly and may introduce undesired interactions that negatively affect our two goals. Thus, we only keep the interactions that are useful to our task and remove the others. Given user u, his preferred attributes in the conversation $\mathcal{P}_u$, and the target item v, we predict how likely u will like v in the conversation session as:

$$\hat{y}(u, v, \mathcal{P}_u) = \mathbf{u}^T\mathbf{v} + \sum_{p_i \in \mathcal{P}_u} \mathbf{v}^T\mathbf{p_i}, \tag{1}$$

where $\mathbf{u}$ and $\mathbf{v}$ denote the embedding for user u and item v, respectively, and $\mathbf{p}_i$ denotes the embedding for attribute $p_i \in \mathcal{P}_u$. Bias terms are omitted for clarity. The first term $\mathbf{u}^T\mathbf{v}$ models the general interest of the user on the target item, a common term in FM model [17]. The second term $\sum \mathbf{v}^T\mathbf{p_i}$ models the affinity between the target item and user preferred attributes. We have also tried to include v's attributes $\mathcal{P}_v$ into FM, but found it brings no benefits. One possible reason is that the item embedding $\mathbf{v}$ may have already encoded its attribute information. Thus we also omit it.

To train the FM, we optimize the pairwise Bayesian Personalized Ranking (BPR) [30] objective. Specifically, given a user u, it assumes the interacted items (e.g., visited restaurants, listened music) should be assigned higher scores than those not interacted with. The loss function of traditional BPR is:

$$L_{bpr} = \sum_{(u,v,v') \in \mathcal{D}_1} -\ln\sigma(\hat{y}(u, v, \mathcal{P}_u) - \hat{y}(u, v', \mathcal{P}_u)) + \lambda_\Theta \|\Theta\|^2 \tag{2}$$

where $\mathcal{D}_1$ is the set of pairwise instances for BPR training, $\mathcal{D}_1 := \{(u, v, v') \mid v' \in \mathcal{V}_u^-\}$, where v is the interacted item of the conversation session (i.e., the ground truth item of the session), $\mathcal{V}_u^- := \mathcal{V} \backslash \mathcal{V}_u^+$ denotes the set of non-interacted items of user u and $\mathcal{V}_u^+$ denotes the items interacted by u. σ is the sigmoid function, and λ_Θ is the regularization parameter to prevent overfitting.

3.1.2 Attribute-aware BPR for Item Prediction. However, in our scenario, the emphasis of CRS is to rank the items that contain the user preferred attributes well. For example, if u specifies "Mexican restaurant" as his preferred attribute, a good CRS needs to rank his preferred restaurants among all available Mexican restaurants. To capture this, we propose to sample two types of negative examples:

$$\mathcal{V}_u^- := \mathcal{V} \backslash \mathcal{V}_u^+, \quad \widehat{\mathcal{V}}_u^- := \mathcal{V}_{cand} \backslash \mathcal{V}_u^+, \tag{3}$$

where $\mathcal{V}_u^-$ is the same negative samples as in the traditional BPR setting, i.e., all non-interacted items of u. $\mathcal{V}_{cand}$ denotes the current candidate items satisfying the partially known preference $\mathcal{P}_u$ in the conversation, and $\widehat{\mathcal{V}}_u^-$ is the subset of $\mathcal{V}_{cand}$ that excludes the observed items $\mathcal{V}_u^+$. The two types of pairwise training instances is defined as:

$$\mathcal{D}_1 := \{(u, v, v') \mid v' \in \mathcal{V}_u^-\}, \quad \mathcal{D}_2 := \{(u, v, v') \mid v' \in \widehat{\mathcal{V}}_u^-\}, \tag{4}$$

We then train the FM model by optimizing both $\mathcal{D}_1$ and $\mathcal{D}_2$:

$$\begin{aligned} L_{item} = &\sum_{(u,v,v') \in \mathcal{D}_1} -\ln\sigma(\hat{y}(u, v, \mathcal{P}_u) - \hat{y}(u, v', \mathcal{P}_u)) \\ &+ \sum_{(u,v,v') \in \mathcal{D}_2} -\ln\sigma(\hat{y}(u, v, \mathcal{P}_u) - \hat{y}(u, v', \mathcal{P}_u)) + \lambda_\Theta \|\Theta\|^2, \end{aligned} \tag{5}$$

where the first loss learns u's general preference, and the second loss learns u's specific preference given the current candidates. It

[2]We detail how to obtain such data in experiments Section 4.1.2.

is worth noting adding the second loss for training is critical for the model ranking well on the current candidates. This is very important for CRS since the candidate items dynamically change with user feedback along the conversation. However, the state-of-the-art method CRM [32] does not account for this factor, being insufficient in considering the interaction between the CC and RC.

3.1.3 Attribute Preference Prediction. We formulate the task of the second goal of accurate attribute prediction separately. This prediction of attribute preference is mainly used in the CC to support the action on which attribute to ask (*c.f.* Sec 3.2). As such, we take u's preferred attributes in the current session into account:

$$\hat{g}(p|u, \mathcal{P}_u) = \mathbf{u}^T \mathbf{p} + \sum_{p_i \in \mathcal{P}_u} \mathbf{p}^T \mathbf{p}_i, \tag{6}$$

which estimates u's preference on attribute p, given u's current preferred attributes $\mathcal{P}_u$. To train the model, we also employ BPR loss, and assume that the attributes of the ground truth item v (of the session) should be ranked higher than other attributes:

$$L_{attr} = \sum_{(u,p,p') \in \mathcal{D}_3} -\ln \sigma(\hat{g}(p|u, \mathcal{P}_u) - \hat{g}(p'|u, \mathcal{P}_u)) + \lambda_\Theta \|\Theta\|^2, \tag{7}$$

where the pairwise training data D_3 is defined as:

$$D_3 = \{(u, p, p') | p \in \mathcal{P}_v, p' \in \mathcal{P} \backslash \mathcal{P}_v\}, \tag{8}$$

where $\mathcal{P}_v$ denotes item v's attributes.

3.1.4 Multi-task Training. We perform joint training on the two tasks of item prediction and attribute prediction, which has the potential of mutual benefits since their parameters are shared. The multi-task training objective is:

$$L = L_{item} + L_{attr}. \tag{9}$$

Specifically, we first train the model with L_{item}. After it converges, we continue optimizing the model using L_{attr}. We iterate the two steps until convergence under both losses. Empirically, 2-3 iterations are sufficient for convergence.

3.2 Action

After the estimation stage, the action stage finds the best strategy for *when to recommend*. We adopt reinforcement learning (RL) to tackle this multi-round decision making problem, aiming to accomplish successful recommendation in shorter number of turns. It is worth noting that since our focus is on conversational recommendation strategy, as opposed to fluent dialogue (the language part), we use templates as wrappers to handle user utterances and system response generation. That is to say, this work serves as an upper bound study of real applications as we do not include the errors for language understanding and generation.

3.2.1 State Vector. The state vector is a bridge for the interaction between the CC and RC. We encode information from the RC and dialogue history into a state vector, providing it to the CC to choose actions. The state vector is a concatenation of four component vectors that encode signal from different perspectives:

$$\mathbf{s} = \mathbf{s}_{ent} \oplus \mathbf{s}_{pre} \oplus \mathbf{s}_{his} \oplus \mathbf{s}_{len}. \tag{10}$$

Each of the vector components captures an assumption on asking which attribute could be most useful, or whether now is a good time to push a recommendation. They are defined as follows:

- $\mathbf{s}_{ent}$: This vector encodes the entropy information of each attribute among the attributes of the current candidate items $\mathcal{V}_{cand}$. The intuition is that asking attributes with large entropy helps to reduce the candidate space, thus benefits finding desired items in fewer turns. Its size is the attribute space size $|\mathcal{P}|$, where the i-th dimension denotes the entropy of the attribute p_i.
- $\mathbf{s}_{pre}$: This vector encodes u's preference on each attribute. It is also of size $|\mathcal{P}|$, where each dimension is evaluated by Equation (6) on the corresponding attribute. The intuition is that the attribute with high predicted preference is likely to receive positive feedback, which also helps to reduce the candidate space.
- $\mathbf{s}_{his}$: This vector encodes the conversation history. Its size is the number of maximum turns T, where each dimension t encodes user feedback at turn t. Specifically, we use -1 to represent recommendation failure, 0 to represent asking an attribute that u disprefers, and 1 to represent successfully asking about an attribute that u desires. This state is useful to determine when to recommend items. For example, if the system has asked about a number of attributes for which u approves, it may be a good time to recommend.
- $\mathbf{s}_{len}$: This vector encodes the length of the current candidate list. The intuition is that if the candidate list is short enough, EAR should turn to recommending to avoid wasting more turns. We divide the length $|\mathcal{V}_{cand}|$ into ten categorical (binary) features to facilitate the RL training.

It is worth noting that besides $\mathbf{s}_{his}$, the other three vectors are all derived from the RC component. We claim that this is a key difference from existing conversational systems [9, 23, 26, 32, 40]; i.e., the CC needs to take information from the RC to decide the dialogue action. In contrast to EAR, the recent conversational recommendation method CRM [32] makes decisions based only on the belief tracker that records the preferred attributes of the user, which makes it less informative. As such, CRM is less effective especially when the number of attributes is large (their experiments only deal with 5 attributes, which is insufficient for real-world applications).

3.2.2 Policy Network and Rewards. The conversation action is chosen by a policy network in our CC. In order to demonstrate the efficacy of our designed state vector, we purposely choose a simple policy network — a two-layer multi-layer perceptron, which can be optimized with the standard policy gradient method. It contains two fully-connected layers and maps the state vector $\mathbf{s}$ into the action space. The output layer is normalized to be a probability distribution over all actions by $softmax$. In terms of the action space, we follow the previous method [32], which includes all attributes $\mathcal{P}$ and a dedicated action for recommendation. To be specific, we define the action space as $\mathcal{A} = \{a_{rec} \cup \{a_{ask}(p) | p \in \mathcal{P}\}\}$, which is of size $|\mathcal{P}| + 1$. After the CC takes an action at each turn, it will receive an immediate reward from the user (or user simulator). This will guide the CC to learn the optimal policy that optimizes long-term reward. In EAR, we design four kinds of rewards, namely: (1) r_{suc}, a strongly positive reward when the recommendation is successful, (2) r_{ask}, a positive reward when the user gives positive feedback on the asked attribute, (3) r_{quit}, a strongly negative reward if the user quits the conversation, (4) r_{prev}, a slightly negative reward

on every turn to discourage overly lengthy conversations. The intermediate reward r_t at turn t is the sum of the above four rewards, $r_t = r_{suc} + r_{ask} + r_{quit} + r_{prev}$.

We denote the policy network as $\pi(a^t \mid \mathbf{s}^t)$, which returns the probability of taking action a^t given the state $\mathbf{s}^t$. Here $a^t \in \mathcal{A}$ and $\mathbf{s}^t$ denote the action to take and the state vector of the t-th turn, respectively. To optimize the policy network, we use the standard policy gradient method [33], formulated as follows:

$$\theta \leftarrow \theta - \alpha \bigtriangledown \log \pi_\theta(a^t \mid \mathbf{s}^t) R_t, \tag{11}$$

where θ denotes the parameter of the policy network, α denotes the learning rate of the policy network, and R_t is the total reward accumulating from turn t to the final turn T: $R_t = \sum_{t'=t}^{T} \gamma^{T-t'} r_{t'}$, where γ is a discount factor which discounts future rewards over immediate reward.

3.3 Reflection

This stage also implements the interaction between the CC and RC. It is triggered when the CC pushes the recommended items $\mathcal{V}^t$ to the user but gets rejected, so as to update the RC model for better recommendations in future turns. In the traditional static recommender system training scenario [17, 30], one issue is the absence of true negative samples, since users do not explicitly indicate what they dislike. In our conversational case, the rejection feedback is an explicit signal on user dislikes which are highly valuable to utilize; moreover, it indicates that the offline learned FM model improperly assigns high scores to the rejected items. To leverage on this source of feedback, we treat the rejected items $\mathcal{V}^t$ as negative samples, constructing more training examples to refresh the FM model. Following the offline training process, we also optimize the BPR loss:

$$L_{ref} = \sum_{(u,v,v') \in \mathcal{D}_4} -\ln \sigma(\hat{y}(u,v,\mathcal{P}_u) - \hat{y}(u,v',\mathcal{P}_u)) + \lambda_\Theta \|\Theta\|^2 \tag{12}$$

where $\mathcal{D}_4 := \{(u,v,v') \mid v \in \mathcal{V}_u^+ \wedge v' \in \mathcal{V}^t\}$. Note that this stage is performed in an online fashion, where we do not have access to the ground truth positive item. Thus, we treat the historically interacted items $\mathcal{V}_u^+$ as the positive items to pair with the rejected items. We put all examples in D_4 into a batch and perform batch gradient descent. Empirically, it takes 3-5 epochs to converge, sufficiently efficient for online use.

Note that although it sounds reasonable to also update the policy network of the CC (since the rejection feedback implies that it is not an appropriate timing to push recommendation), we currently do not perform this due to high difficulty of online updating RL agent and leave it for future work.

4 EXPERIMENTS

EAR [3] is built based on the guiding ideology of interaction between the CC and RC. To validate this ideology, we first evaluate the whole system to examine the overall effect brought by the interaction. Then, we perform ablation study to investigate the effect of interaction on each individual component. Specifically, we have the following research questions (RQ) to guide experiments on two datasets.

- **RQ1.** How is the overall performance of EAR comparing with existing conversational recommendation methods?
- **RQ2.** How do the attribute-aware BPR and multi-task training of the *estimation* stage contribute to the RC?
- **RQ3.** Is the state vector designed for the CC in the *action* stage appropriate?
- **RQ4.** Is the online model update of the *reflection* stage useful in obtaining better recommendation?

Table 1: Dataset statistics.

Dataset	#users	#items	#interactions	#attributes
Yelp	27,675	70,311	1,368,606	590
LastFM	1,801	7,432	76,693	33

4.1 Settings

4.1.1 Datasets. We conduct experiments on two datasets: Yelp[4] for business recommendation and LastFM[5] for music artist recommendation. First, we follow the common setting of recommendation evaluation [17, 30] that reduces the data sparsity by pruning the users that have less than 10 reviews. We split the user–item interactions in the ratio of 7:2:1 for training, validation and testing. Table 1 summarizes the statistics of the datasets.

For the item attributes, we preprocess the original attributes of the datasets by merging synonyms and eliminating low frequency attributes, resulting in 590 attributes in Yelp and 33 attributes in LastFM. In real applications, asking about attributes in a large attribute space (e.g., on Yelp dataset) causes overly lengthy conversation. We therefore consider both the binary question setting (on LastFM) and enumerated question (on Yelp). To enable the enumerated question setting, we build a two-level taxonomy on the attributes of the Yelp data. For example, the *parent attribute* of {"wine", "beer", "whiskey"} is "alcohol". We create 29 such parent attributes on the top of the 590 attributes, such as "nightlife", "event planning services", "dessert types" etc. In the enumerated question setting, the system choose one parent attribute to ask. This is to say, we change the size of the output space of the policy network to be 29 + 1 = 30. At the same time, it also displays all its child attributes and ask the user to choose from them (the user can reply with multiple child attributes). Note that choosing what kinds of questions to ask is an engineering design choice by participants, here we evaluate our model on both settings.

4.1.2 User Simulator For Multi-round Scenario. Because the conversational recommendation is a dynamic process, we follow [32, 40]) to create a user simulator to enable the CRS training and evaluation. We simulate a conversation session for each observed interaction between users and items. Specifically, given an observed user–item interaction (u, v), we treat the v as the ground truth item to seek for and its attributes $\mathcal{P}_v$ as the oracle set of attributes preferred by the user in this session. At the beginning, we randomly choose an attribute from the oracle set as the user's initialization to the

[3] Datasets, source code and demos at our project homepage: https://ear-conv-rec.github.io

[4] https://www.yelp.com/dataset/

[5] https://grouplens.org/datasets/hetrec-2011/

session. Then the session goes in the loop of the "model acts – simulator response" process as introduced in Section 2. We set the max turn T of a session to 15 and standardize the recommendation list length $\mathcal{V}^t$ as 10.

4.1.3 Training Details. Following CRM [32], the training process is divided into offline and online stages. The offline training is to build the RC (i.e., FM) and initialize the policy network (PN) by letting them optimize performance with the offline dialogue history. Due to the scarcity of the conversational recommendation dialogue history, we follow CRM [32] to simulate dialogue history by building a rule-based CRS to interact with the simulator introduced in Section 4.1.2. Specifically, the strategy for determining which attribute to ask about is to choose the attribute with the maximum entropy. Each turn, the system chooses the recommendation action with probability $10/max(|\mathcal{V}|, 10)$ where $\mathcal{V}$ is the current candidate set. The intuition is that the confidence of recommendation grows when the candidate size is smaller. We train the RC to give the ground-truth item and oracle attributes higher ranks given the attribute confirmed by users in dialogue histories, while training the policy to mimic the rule-based strategy on the history. Afterwards, we conduct online training, optimizing the PN by letting EAR interact with the user simulator through reinforcement learning.

We tuned all hyper-parameters on the validation set, and empirically set them as followed: The embedding size of FM is set as 64. We employ the multi-task training mechanism to optimize FM as described in Section 3.1.4, using SGD with a regularization strength of 0.001. The learning rate for the first task (item prediction) and second task (attribute prediction) is set to 0.01 and 0.001, respectively. The size of the two hidden layers in the PN is set as 64. When the pre-trained model is initialized, we use the REINFORCE algorithm to train the PN. The four rewards are set as: r_{suc}=1, r_{ask}=0.1, r_{quit}=-0.3, and r_{prev}=-0.1, and the learning rate α is set as 0.001. The discount factor γ is set to be 0.7.

4.1.4 Baselines. As our multi-round conversational recommendation scenario is new, there are few suitable baselines. We compare our overall performance with the following three:

- **Max Entropy**. This method follows the rule we used to generate the conversation history in Section 4.1.2. Each turn it asks the attribute that has the maximum entropy among the candidate items. It is claimed in [12] that maximum entropy is the best strategy when language understanding is precise. It's worth noting that, in enumerated question setting, the entropy of an attribute is calculated as the sum of its child attributes in the taxonomy (similar approach for attribute preference calculation).
- **Abs Greedy** [10]. This method recommends items in every turn without asking any question. Once the recommendation is rejected, it updates the model by treating the rejected items as negative examples. According to [10], this method achieves equivalent or better performance than popular bandit algorithms like Upper Confidence Bounds [1] and Thompson Sampling [4].
- **CRM** [32]. This is a state-of-the-art CRS. Similar to EAR, it integrates a CC and RC by feeding the belief tracker results to FM for item prediction, without considering much interactions between them. It is originally designed for single-round recommendation. To achieve a fair comparison, we adapt it to the multi-round setting by following the same offline and online training of EAR.

It is worth noting that although there are other recent conversational recommendation methods [10, 23, 26, 40], they are ill-suited for comparison due to their different task settings. For example, [40] focuses on document representation which is unnecessary in our case. It also lacks the conversation policy component to decide when to make what action. [23] focuses more on language understanding and generation. We summarize the settings of these methods in Table 6 and discuss differences in Section 5.

Table 2: SR@15 and AT of compared methods. * denotes that improvement of EAR over other methods is statistically significant for $p < 0.01$ (RQ1).

	LastFM		Yelp	
	SR@15	**AT**	**SR@15**	**AT**
Abs Greedy	0.209	13.63	0.271	12.26
Max Entropy	0.290	13.61	0.919	5.77
CRM	0.325	13.43	0.923	5.33
EAR	**0.429***	**12.45***	**0.971***	**4.71***

4.1.5 Evaluation Metrics. We use the success rate (SR@t) [32] to measure the ratio of successful conversations, i.e., recommend the ground truth item by turn t. We also report the average turns (AT) needed to end the session. Larger SR denotes better recommendation and smaller AT denotes more efficient conversation. When studying RC model of offline training, we use the AUC score which is a surrogate of the BPR objective [30]. We conduct one-sample paired t-test to judge statistical significance.

4.2 Performance Comparison (RQ1)

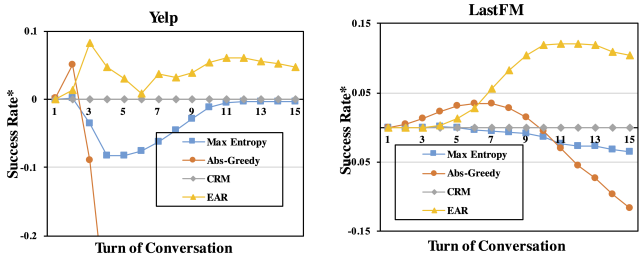

Figure 2: Success Rate* of compared methods at different conversation turns on Yelp and LastFM (RQ1).

Figure 2 shows the recommendation Success Rate* (SR*) @t at different turns (t = 1 to 15), SR* denotes the comparison of each method against the strongest baseline CRM, indicated as $y = 0$ in the figure. Table 2 shows the scores of the final success rate and the average turns. As can be clearly seen, our EAR model significantly outperforms other methods. This validates our hypothesis that considering extensive interactions between the CC and RC is an effective strategy to build conversational a recommender system. We also make the following observations:

Comparing with Abs Greedy, the three attribute-based methods (EAR, Max Entropy and CRM) have nearly zero success rate at the beginning of a conversation ($t < 2$). This is because these methods tend to ask questions at the very beginning. As the conversation goes, Abs Greedy (which only recommends items) gradually

falls behind the attribute-based methods, demonstrating the efficacy of asking attributes in the conversational recommendation scenario. Note that Abs Greedy has much weaker performance on Yelp compared to LastFM. The key reason is the setting of Yelp is to ask enumerated question, and user's response with multiple finer-grained attributes sharply shrinks the candidate items.

CRM generally underperforms our EAR methods. One of the key reasons is that its state vector cannot help CC to learn sophisticated strategy to ask and recommend, especially in a much larger action space, i.e., the number of attributes (nearly 30 in our experiments versus 5 in theirs [32]). This result suggests that in a more complex *multi-round* scenario where the CC needs to make a comprehensive utilization of both the CC (e.g., considering dialogue histories) and RC (considering statistics like attribute preference estimation) when formulating a recommendation strategy.

Interestingly, Figure 2 indicates that in Yelp, EAR's gain over CRM enlarges in Turns 1–3, shrinks in Turns 4–6 and widens again afterwards. However, in LastFM it has a steadily increasing gain. This interesting phenomenon reveals that our EAR system can learn different strategies in different settings. In the Yelp dataset, the CRS asks enumerated questions where the user can choose finer-grained attributes, resulting a sharp reduction in the candidate space. The strategy that the EAR system learns is more aggressive: it attempts to ask attributes that can sharply shrink the candidate space and make decisive recommendation at the beginning turns when it feels confident. If this aggressive strategy fails, it changes to a more patient strategy to ask more questions without recommendations, causing less success in the medial turns (e.g., Turns 5–7). However, this strategy pays off in the long term, making recommendation more successful in the latter half of conversations (e.g., after Turn 7). At the same time, CRM is only able to follow the strategy of trying to ask more attributes at the beginning and making recommendations later. In the LastFM dataset, the setting is limited to binary attributes, leading to less efficiency in reducing candidate space. Both EAR and CRM adapt and ask more questions at the outset before making recommendations. However, as EAR incorporates better CC and RC to model better interaction, it significantly outperforms CRM.

4.3 Effectiveness of Estimation Designs (RQ2)

There are two key designs in the estimation stage that trains the recommendation model FM offline: the attribute-aware BPR that samples negatives with attribute matching considered, and the multi-task training that jointly optimizes item prediction and attribute prediction tasks. Table 3 shows offline AUC scores on the two tasks of three methods: FM, FM with attribute-aware BPR (FM+A), and FM+A with multi-task training (FM+A+MT).

As can be seen, the attribute-aware BPR significantly boosts the performance of item ranking, being highly beneficial to rank the ground truth item high. Interestingly, it harms the performance of attribute prediction, e.g. on lastFM, FM+A has a much lower AUC score (0.629) than FM (0.727). The reason might be that the attribute-aware BPR loss guides the model to specifically fit to item ranking in the candidate list. Without an even optimization enforced for the attribute prediction task, it may suffer from poor performance. This implies the necessity of explicitly optimizing the attribute prediction task. As expected, the best performance is achieved when we add multi-task training on. FM+A+MT significantly enhances the performance of both tasks, validating the effectiveness and rationality of our multi-task training design.

Table 3: Offline AUC score of FM, FM with attribute-aware BPR (FM+A) and with multi-task training for item recommendation and attribute prediction (FM+A+MT). * denotes that improvement of FM+A+MT over FM and FM+A is statistically significant for $p < 0.01$ (RQ2).

	LastFM		Yelp	
	Item	**Attribute**	**Item**	**Attribute**
FM	0.521	0.727	0.834	0.654
FM+A	0.724	0.629	0.866	0.638
FM+A+MT	**0.742***	**0.760***	**0.870***	**0.896***

Table 4: Performance of removing one component of the state vector (Equation 10) from our EAR. * denotes that improvement of EAR over model with removed component is statistically significant for $p < 0.01$ (RQ 3).

	Yelp				LastFM			
	SR@5	SR@10	SR@15	AT	SR@5	SR@10	SR@15	AT
$-\mathbf{s}_{ent}$	0.614	0.895	0.969	4.81	**0.051**	0.190	0.346	12.82
$-\mathbf{s}_{pre}$	0.596	0.857	0.959	5.06	0.024	0.231	0.407	12.55
$-\mathbf{s}_{his}$	0.624	0.894	0.949	4.79	0.021	0.236	0.424	12.50
$-\mathbf{s}_{len}$	0.550	0.846	0.952	5.44	0.013	0.230	0.416	12.56
EAR	**0.629***	**0.907***	**0.971***	**4.71***	0.020	**0.243***	**0.429***	**12.45***

4.4 Ablation Studies on State Vector (RQ3)

What information helps in decision making? Let us examine the effects of the the four forms of information included in EAR state vector $\mathbf{s}$ (Equation 10), by ablating each information type from the feature vector (Table 4).

Comparing the performance drop of each method, we uncover differences that corroborate the intrinsic difference between the two conversational settings. The most important factor is question type: i.e., $\mathbf{s}_{ent}$ for LastFM (binary question) and $\mathbf{s}_{len}$ for Yelp (enumerated question). The entropy($\mathbf{s}_{ent}$) information is crucial for LastFM, it is in line with the claim in [12] that the maximum entropy is the best strategy when language understanding is precise. If we ablate $\mathbf{s}_{ent}$ on LastFM, although it reaches 0.051 in SR@5, future SR greatly suffers, due to the system's over-agressiveness to recommend items before obtaining sufficient relevant attribute evidence. As for the enumerated question setting (Yelp), the candidate list length ($\mathbf{s}_{len}$) is most important, because the candidate item list shrinks more sharply and $\mathbf{s}_{len}$ is helpful when deciding when to recommend.

Apart from entropy and candidate list length, the remaining two factors – i.e., attribute preference, conversation history – both contribute positively. Their impact is sensitive to datasets and metrics. For example, the attribute preference ($\mathbf{s}_{pre}$) strongly affect SR@5 and SR@10 on Yelp, but does not show significant impacts for SR@15. This inconsistency provides an evidence for the intrinsic difficulty of decision making in the conversational recommendation scenario, which however has yet to be extensively studied.

4.5 Investigation on Reflection (RQ4)

To understand the impact of online update in the reflection stage, we start from the ablation study. Table 5 shows the variant of EAR

Table 5: Performance after removing the online update module in the reflection stage. * denotes that improvement of EAR over removing update module is statistically significant for $p < 0.01$ (RQ4).

	Yelp				LastFM			
	SR@5	SR@10	SR@15	AT	SR@5	SR@10	SR@15	AT
-update	0.629	0.905	0.970	4.72	0.020	0.217	0.393	12.67
EAR	**0.629**	**0.907**	**0.971**	**4.71**	**0.020**	**0.243***	**0.429***	**12.45***

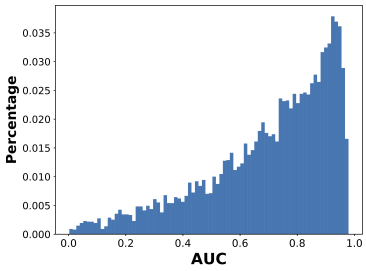

Figure 3: Percentage of bad updates w.r.t. the offline model's AUC on the users on Yelp (RQ4).

that removes online update. We find that the trends do not converge on two datasets: the updating strategy helps a lot on LastFM but has very minor effect on the Yelp dataset.

Questioning this interesting phenomenon, we examine the individual items on Yelp. We find that the updating does not always help ranking, especially when the offline model already ranks the ground truth item high (but not at top 10). In this case, doing updates is highly likely to pull down the ranking position of the ground truth item. To gain statistical evidence for this observation, we term such updates as *bad updates*, and show the percentage of bad updates with respect to the offline model's AUC on the users. As seen from Figure 3, there is a clear positive correlation between bad updates and AUC score. For example, ~3.5% of the bad updates come from users with an offline AUC of 0.9.

This explains why online update works well for LastFM, but not for Yelp: our recommendation model has a better performance on Yelp than LastFM (0.870 v.s. 0.742 in AUC as shown in Table 3). This means the items on Yelp are more likely to get higher AUC, resulting in worse updates. More such observations and analyses will help further the community understanding the efficacy of online updates. Although bandit algorithms have devoted to exploring this question [11, 14, 21, 24, 37], the issue has largely been unaddressed in the context of conversational recommender system.

5 RELATED WORK

The offline **static recommendation** task is formulated as estimating the affinity score between a user and an item [17]. This is usually achieved by learning user preferences through the historical user-item interactions such as clicking and purchasing. The representative methods are Matrix Factorization (MF) [20] and Factorization Machine (FM) [29]. Neural FM [16] and DeepFM [15] have improved FM's representation ability with deep neural networks. [3, 13, 18] utilize user's implicit feedback, commonly optimizing BPR loss [30]. [7, 8] exploits user's reviews and image information. However, such static recommendation methods suffer from the intrinsic limitation of not being able to capture user dynamic preferences.

This intrinsic limitation motivates **online recommendation**. Its target is to adapt the recommendation results with the user's online actions [25]. Many model it as a multi-arm bandit problem [34, 36, 37] , strategically demonstrating items to users for useful feedback. [39] makes the preliminary effort to extend the bandit framework to query attributes. While achieving remarkable progress, the bandit-based solutions are still insufficient: 1) Such methods focus on exploration–exploitation trade-off in cold start settings. However, in warm start scenario, capturing the user dynamic preference is critical as preference drift is common; 2) The mathematical formation of multi-arm bandit problem limits such method only recommend one item each time. This constraint limits its application, as we usually need to recommend a list of items.

Conversational recommender systems provide a new possibility for capturing dynamic feedback as they enable a system to interact with users using natural language. However, they also pose challenges to researchers, leading to various settings and problem formulations [2, 9, 10, 23, 26–28, 31, 32, 38–40]. Table 6 summarizes these works' key aspects. Generally, prior work considers conversational recommendation only under simplified settings. For example, [10, 38] only allow the CRS to recommend items without asking the user about their preferred attributes. The Q&R work [9] proposes to jointly optimize the two tasks of attribute and item prediction, but restricts the whole conversation to two turns: one turn for asking, one turn for recommending. CRM [32] extends the conversation to multi-turns but still follows the single-round setting. MMN [40] focuses on document representation, aiming to learn better matching function for attributes and products description under a conversation setting. Unfortunately, it does not build a dialogue policy to decide when to ask or make recommendations. In contrast, situations for various real applications are complex: the CRS needs to strategically ask attributes and make recommendations in multiple rounds, achieving successful recommendations in the fewest turns. In recent work, only [23] considers this multi-round scenario, but it focuses on language understanding and generation, without attending to explicitly model the conversational strategy.

6 CONCLUSION AND FUTURE WORK

In this work, we redefine the conversational recommendation task where the RC and CC closely support each other so as to achieve the goal of accurate recommendation in fewer turns. We decompose the task into three key problems, namely, what to ask, when to recommend, and how to adapt with user feedback. We then propose EAR – a new three-stage solution accounting for the three problems in a unified framework. For each stage, we design our method to carefully account for the interactions between RC and CC. Through extensive experiments on two datasets, we justify the effectiveness of EAR, providing additional insights into the conversational strategy and online updates.

Our work represents the first step towards exploring how the CC and RC can collaborate closely to provide quality recommendation service in this multi-round scenario. Naturally, there are thus a few loose ends for further investigation, especially with respect to incorporating user feedback. In the future, we will consider

Table 6: Recent conversational recommender summary: 1) whether it asks about attributes, 2) question space, 3) any explicit strategy w.r.t. recommendation timing, 4) whether it considers multi-round recommendations, and 5) its main focus.

	1. Q?	2. Question Space	3. Explicit	4. Multi-round	5. Main Focus
Online bandits [10, 36, 37]	×	N.A.	×	✓	Exploration-exploitation trade-off in item selection
REDIAL (NIPS'18) [23]	✓	Free texts	×	✓	End-to-end generation of natural language response
KMD (MM'18) [26]	✓	Free texts	×	✓	End-to-end generation of text and image response
Q&R (KDD'18) [9]	✓	Attributes	×	×	Question asking and single-round recommendation
MMN (CIKM'18) [40]	✓	Attributes	×	✓	Attribute-product match in conversational search
CRM (SIGIR'18) [32]	✓	Attributes	✓	×	Shallow combination between CC and RC
VDARIS (KDD'19) [38]	×	N.A.	×	✓	User's click and comment on recommended items
EAR (our method)	✓	Attributes	✓	✓	Deep interaction between CC and RC

refreshing the policy network to make better actions. We will also extend EAR to consider explore–exploit balance which is the key problem for traditional interactive recommendation system. Lastly, we will deploy our system to online applications that interact with real users to gain more insights for further improvements.

Acknowledgement: This research is part of NExT++ research and also supported by the National Natural Science Foundation of China (61972372). NExT++ is supported by the National Research Foundation, Prime Minister's Office, Singapore under its IRC@SG Funding Initiative. We would like to thank the anonymous reviewers for their valuable reviews.

REFERENCES

[1] Peter Auer. 2002. Using confidence bounds for exploitation-exploration trade-offs. *Journal of Machine Learning Research* 3, Nov (2002), 397–422.
[2] MS Ayundhita, ZKA Baizal, and Y Sibaroni. 2019. Ontology-based conversational recommender system for recommending laptop. In *Journal of Physics: Conference Series*, Vol. 1192. IOP Publishing, 012020.
[3] Immanuel Bayer, Xiangnan He, Bhargav Kanagal, and Steffen Rendle. 2017. A generic coordinate descent framework for learning from implicit feedback. In *WWW*. 1341–1350.
[4] Olivier Chapelle and Lihong Li. 2011. An empirical evaluation of thompson sampling. In *NeurIPS*. 2249–2257.
[5] Hongshen Chen, Zhaochun Ren, Jiliang Tang, Yihong Eric Zhao, and Dawei Yin. 2018. Hierarchical Variational Memory Network for Dialogue Generation. In *WWW*. 1653–1662.
[6] Jingyuan Chen, Hanwang Zhang, Xiangnan He, Liqiang Nie, Wei Liu, and Tat-Seng Chua. 2017. Attentive Collaborative Filtering: Multimedia Recommendation with Item- and Component-Level Attention. In *SIGIR*. 335–344.
[7] Zhiyong Cheng, Xiaojun Chang, Lei Zhu, Rose C Kanjirathinkal, and Mohan Kankanhalli. 2019. MMALFM: Explainable recommendation by leveraging reviews and images. *TOIS* 37, 2 (2019), 16.
[8] Zhiyong Cheng, Ying Ding, Lei Zhu, and Mohan Kankanhalli. 2018. Aspect-aware latent factor model: Rating prediction with ratings and reviews. In *Proceedings of the 2018 World Wide Web Conference*. International World Wide Web Conferences Steering Committee, 639–648.
[9] Konstantina Christakopoulou, Alex Beutel, Rui Li, Sagar Jain, and Ed H Chi. 2018. Q&R: A Two-Stage Approach toward Interactive Recommendation. In *SIGKDD*. 139–148.
[10] Konstantina Christakopoulou, Filip Radlinski, and Katja Hofmann. 2016. Towards conversational recommender systems. In *SIGKDD*. 815–824.
[11] Wei Chu, Lihong Li, Lev Reyzin, and Robert Schapire. 2011. Contextual bandits with linear payoff functions. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. 208–214.
[12] Bhuwan Dhingra, Lihong Li, Xiujun Li, Jianfeng Gao, Yun-Nung Chen, Faisal Ahmed, and Li Deng. 2017. Towards End-to-End Reinforcement Learning of Dialogue Agents for Information Access. In *ACL*. 484–495.
[13] Travis Ebesu, Bin Shen, and Yi Fang. 2018. Collaborative Memory Network for Recommendation Systems. In *SIGIR*. 515–524.
[14] Claudio Gentile, Shuai Li, and Giovanni Zappella. 2014. Online clustering of bandits. In *ICML*. 757–765.
[15] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. 2017. Deepfm: a factorization-machine based neural network for ctr prediction. In *IJCAI*.
[16] Xiangnan He and Tat-Seng Chua. 2017. Neural factorization machines for sparse predictive analytics. In *SIGIR*. 355–364.
[17] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural Collaborative Filtering. In *WWW*. 173–182.
[18] Xiangnan He, Hanwang Zhang, Min-Yen Kan, and Tat-Seng Chua. 2016. Fast matrix factorization for online recommendation with implicit feedback. In *SIGIR*. 549–558.
[19] Xisen Jin, Wenqiang Lei, Zhaochun Ren, Hongshen Chen, Shangsong Liang, Yihong Zhao, and Dawei Yin. 2018. Explicit State Tracking with Semi-Supervisionfor Neural Dialogue Generation. In *CIKM*. ACM, 1403–1412.
[20] Yehuda Koren, Robert M. Bell, and Chris Volinsky. 2009. Matrix Factorization Techniques for Recommender Systems. *IEEE Computer* 42, 8 (2009), 30–37.
[21] Volodymyr Kuleshov and Doina Precup. 2014. Algorithms for multi-armed bandit problems. *arXiv preprint arXiv:1402.6028* (2014).
[22] Wenqiang Lei, Xisen Jin, Min-Yen Kan, Zhaochun Ren, Xiangnan He, and Dawei Yin. 2018. Sequicity: Simplifying Task-oriented Dialogue Systems with Single Sequence-to-Sequence Architectures. In *ACL*. 1437–1447.
[23] Raymond Li, Samira Ebrahimi Kahou, Hannes Schulz, Vincent Michalski, Laurent Charlin, and Chris Pal. 2018. Towards Deep Conversational Recommendations. In *NeurIPS*. 9748–9758.
[24] Shuai Li, Alexandros Karatzoglou, and Claudio Gentile. 2016. Collaborative filtering bandits. In *SIGIR*. 539–548.
[25] Seth Siyuan Li and Elena Karahanna. 2015. Online recommendation systems in a B2C E-commerce context: a review and future directions. *Journal of the Association for Information Systems* 16, 2 (2015), 72.
[26] Lizi Liao, Yunshan Ma, Xiangnan He, Richang Hong, and Tat-Seng Chua. 2018. Knowledge-aware Multimodal Dialogue Systems. In *ACM MM*. 801–809.
[27] Lizi Liao, Ryuichi Takanobu, Yunshan Ma, Xun Yang, Minlie Huang, and Tat-Seng Chua. 2019. Deep Conversational Recommender in Travel. *arXiv preprint arXiv:1907.00710* (2019).
[28] Bilih Priyogi. 2019. Preference Elicitation Strategy for Conversational Recommender System. In *WSDM*. ACM, 824–825.
[29] Steffen Rendle. 2010. Factorization machines. In *ICDM*. IEEE, 995–1000.
[30] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian personalized ranking from implicit feedback. In *UAI*.
[31] Nicola Sardella, Claudio Biancalana, Alessandro Micarelli, and Giuseppe Sansonetti. 2019. An Approach to Conversational Recommendation of Restaurants. In *ICHCI*. Springer, 123–130.
[32] Yueming Sun and Yi Zhang. 2018. Conversational Recommender System. In *SIGIR*. 235–244.
[33] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. 2000. Policy gradient methods for reinforcement learning with function approximation. In *NeurIPS*. 1057–1063.
[34] Huazheng Wang, Qingyun Wu, and Hongning Wang. 2017. Factorization Bandits for Interactive Recommendation.. In *AAAI*. 2695–2702.
[35] Wenjie Wang, Minlie Huang, Xin-Shun Xu, Fumin Shen, and Liqiang Nie. 2018. Chat More: Deepening and Widening the Chatting Topic via A Deep Model. In *SIGIR*. 255–264.
[36] Qingyun Wu, Naveen Iyer, and Hongning Wang. 2018. Learning Contextual Bandits in a Non-stationary Environment. In *SIGIR*. 495–504.
[37] Qingyun Wu, Huazheng Wang, Quanquan Gu, and Hongning Wang. 2016. Contextual bandits in a collaborative environment. In *SIGIR*. ACM, 529–538.
[38] Tong Yu, Yilin Shen, and Hongxia Jin. 2019. An Visual Dialog Augmented Interactive Recommender System. In *SIGKDD*. ACM, 157–165.
[39] Xiaoying Zhang, Hong Xie, Hang Li, and John Lui. 2019. Toward Building Conversational Recommender Systems: A Contextual Bandit Approach. *arXiv preprint arXiv:1906.01219* (2019).
[40] Yongfeng Zhang, Xu Chen, Qingyao Ai, Liu Yang, and W Bruce Croft. 2018. Towards conversational search and recommendation: System ask, user respond. In *CIKM*. 177–186.