Master of Science in Computer Science

FACT-CRS : Rethinking Conversational Recommendations; Is Decision Tree All you need?

Submitted by

A S M Ahsan-Ul Haque
(ah3wj@virginia.edu)

Supervised by

Dr. Hongning Wang

Department of Computer Science

School of Engineering and Applied Science
University of Virginia

Charlottesville, VA, USA

May 2022

Office of Graduate Programs
School of Engineering and Applied Science
Thornton Hall A108

202004228

Date (mm/dd/yyyy) 4/11/2022

Report on Dissertation or Thesis Final Examination

Student Name (last, first middle) Haque,A S M Ahsan-Ul | Comp ID (e.g. mst3k) ah3wj | SIS ID (office use) 2815241

Above student has submitted in partial fulfillment of the requirements for:

Degree Plan MS in Program Computer Science a dissertation/thesis entitled:

Dissertation/Thesis Title ***(this should be the last, final, and correct title)***

Rethinking Conversational Recommendations: Is Decision Tree All You Need?

The committee examined the candidate on 04/07/2022 (date) in accordance with the regulations governing the Final examination and has judged the candidate's performance to be (X) **Satisfactory** ( ) **Unsatisfactory** and **all program-specific requirements** for the degree have been satisfied or will be satisfied with the completion of course(s) for which the candidate is currently registered. Exceptions or qualification are noted as follows:

Exceptions/Qualifications

Committee Chair

Name	Department/ School/ Institution*	Signature
John Stankovic	CS/SEAS/UVA	DocuSigned by: John Stankovic C67F96E9CE0640F...

Committee Members

Name	Department/ School/ Institution*	Signature
Hongning Wang (Advisor)	CS/SEAS/UVA	DocuSigned by: Hongning Wang 5422E81C963648B...
Tariq Iqbal	ESE/SEAS/UVA	DocuSigned by: Tariq Iqbal CA69F9ACDECD4FF...
-	-	

* School or Institution if not School of Engineering

No fewer than 3 faculty members shall be present for a master's examination, and no fewer than 5 faculty members shall be present for a PhD exam.

Program Approval: DocuSigned by: Felix Lin 0E747249FACE4AC...

Office of Graduate Programs:

M.S. and Ph.D. candidates MUST submit an **Engineering Thesis & Dissertation Assessment form** with this report to the Office of Graduate Programs.
Ph.D. candidates MUST ALSO submit a certificate of completion of the *Survey of Earned Doctorates* (**https://sed.norc.org/doctorate/showRegister.do**) with this report.

CERTIFICATION

This thesis titled, **"FACT-CRS : Rethinking Conversational Recommendations; Is Decision Tree All you need?"**, submitted by the candidate as mentioned below has been accepted as satisfactory in partial fulfillment of the requirements for the degree Master of Science in Computer Science in May 2022.

Candidate:

A S M Ahsan-Ul Haque

Advisor:

Dr. Hongning Wang
Department of Computer Science
School of Engineering and Applied Science
University of Virginia

Committee Members:

Dr. Tariq Iqbal
Department of Engineering Systems and Environment
School of Engineering and Applied Science
University of Virginia

Dr. John Stankovic
Department of Computer Science
School of Engineering and Applied Science
University of Virginia

CANDIDATES' DECLARATION

This is to certify that the work presented in this thesis, titled, "FACT-CRS : Rethinking Conversational Recommendations; Is Decision Tree All you need?", is the outcome of the investigation and research carried out by the candidate under the supervision of Dr. Hongning Wang.

It is also declared that neither this thesis nor any part thereof has been submitted anywhere else for the award of any degree, diploma or other qualifications.

A S M Ahsan-Ul Haque
(ah3wj@virginia.edu)

ABSTRACT

Conversation Recommendation System (CRS) is becoming a topic of interest in information retrieval. Traditional recommender systems rely on historical user interaction records and do not consider the users' current preferences. Alternatively, conversational recommender systems dynamically obtain the users' preferences via multi-turn question and answer. The existing works in conversational recommender systems mostly rely on reinforcement policy learning. Also, the current methods use pre-trained user embeddings for recommendations. However, this approach becomes ineffective when a new user enters the system (the cold start problem). This reduces the effectiveness of CRS for cold-start users, even though it is the main motivation for CRS in the first place.

In this study, we propose two things. Firstly, we challenge the necessity of using reinforcement learning in CRS. There are 3 main challenges in multi-turn CRS: 1) What questions to ask 2) When to recommend 3) How to improve when a user rejects recommendations. We show that supervised machine learning is sufficient to face these 3 challenges. Our supervised learning model is a simple decision tree-based model, namely FACT-CRS (stands for Factorization Tree-based Conversational Recommender System).

Secondly, we show that by taking the user-item interaction into account, we can learn a rule-based method that is effective for users who are new to the system. Extensive experiments on two benchmark CRS datasets (LastFM and Yelp) show that supervised learning is, in fact, sufficient to face the challenges in multi-turn CRS. On the LastFM dataset, our model is 27.42% more successful in recommending the desired item than the best of the baselines. Our model also achieves a 2.05% better success rate than the baselines on the Yelp dataset, which includes more than 1M interactions. Empirical results also suggest that our proposed model can successfully recommend target items to users by asking a fewer number of questions.

To my mother, my father and my sister.

ACKNOWLEDGEMENT

I would like to take the opportunity to thank all the people whose contribution made it possible for me to complete this thesis. Firstly, I would like to express my heartiest gratitude to my supervisor Dr. Hongning Wang for his excellent guidance and overwhelming support. He helped me all the way in this research direction and connected me with a lot of other people whose contribution was vital for this thesis. His perseverance and enthusiasm guided me toward the results achieved in this thesis.

I would also like to express my sincere gratitude the current students at Human Centered Data Mining Group at the University of Virginia (UVA), especially to Yiling Jia, Amar Kulkarni and Zhendong Chu, for their help with various experiments, and for very insightful discussions. I am thankful to Dr. Anil Vullikanti and Dr. Achla Marathe at the UVA Biocomplexity Institute for their support in the beginning of my journey at the University of Virginia. I am thankful to Heath Carelock at PGCC and Dustin Ciraco at the UVA Student Financial Services for their counsel. I am also thankful to Elizabeth Ramirez-Weaver, Mark Silvestri, Sean Sembrowich, and Karen Lai Painter at UVA Counseling and Psychological Services (CAPS) for their earnest support. I am sincerely grateful to Dr. Elizabeth Robinson at the UVA Student Health. I am grateful to Caren Freeman Wendy at the UVA International Studies Office (ISO) for her guidance. I am thankful to Heather Palmer at the UVA University Career Services for her academic and career advice.

I am especially grateful to my friends Jacob Dineen and Md. Fazlay Rabbi Masum Billah, students at the University of Virginia, for their continuous support and guidance.

I would like to sincerely thank Dr. John Stankovic and Dr. Tariq Iqbal for taking the time to serve on the committee. Finally, I would like to thank the University of Virginia for giving me the suitable environment and equipment necessary for the completion of this thesis.

Charlottesville
Virginia
May 2022

A S M Ahsan-Ul Haque

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

It is estimated that we generate billions of gigabytes of data everyday [1]. With the ever increasing amount of generated data and the limitation in our ability to absorb data, we tend to rely on recommendations to look for our desired items. Recommender systems (RS) have become increasingly popular choice for seeking information and items. In this context, the definition of items is very broad and it can be any object that the user is looking for; including products, movies, songs, restaurants, shops etc. Recommender systems have become a ubiquitous phenomenon. For example: product recommendations in e-commerce (e.g. Amazon, Walmart etc.); movies recommendation in streaming services (e.g. Netflix, Hulu etc.), photo/video recommendations (e.g. Facebook, Instagram etc.). All aim to suggest items to the users that are most similar to what the user prefers.

We say a recommendation is effective when it is accurate and it aims to minimize the number of input from the user. An effective recommendation can greatly save time, increase customer satisfaction. Conversely, for a business, it can open the door to new users. It is also very helpful in academic research, since researchers can get suggestions for news and articles their relevant area of research. Naturally, recommender systems have gained a lot of attention in both industry and academic research.

Traditional recommender systems try to score the user's preference solely based on historical user-item interaction data, e.g. explicit rating, text reviews/ comments, clicks, search/ browsing history etc. The key idea behind this kind of approach is the preference of a user for a new item is similar to another user's preference of that new item if both of them preferred other items similarly. Even though many recent traditional approaches have been successfully deployed in a number of real world cases [2–4], they are static as they do not take into account the user's current preference. For example: a user who usually likes a particular type of food (e.g. Mexican food) may want to try another type of food (e.g. Thai food). Also, the user's preference can deviate over time from the historical data.

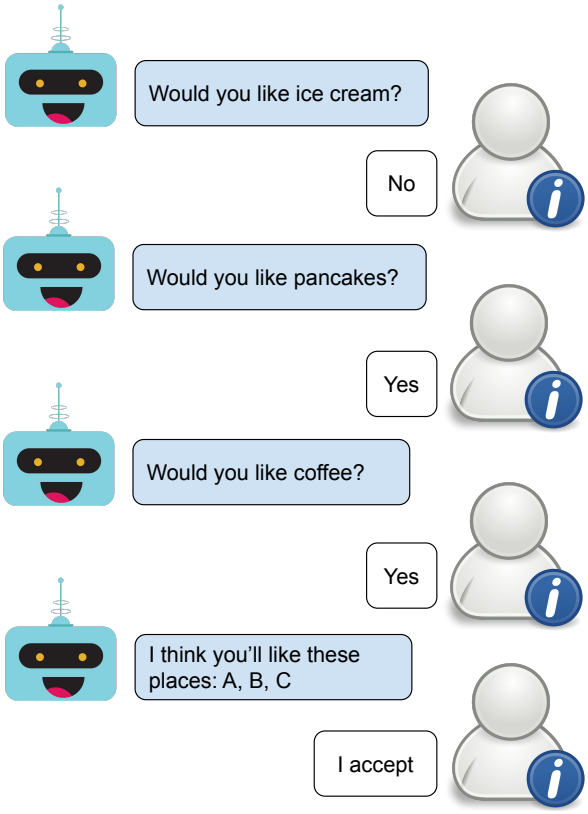

Figure 1.1: Approach in conversational recommender system

In contrast, conversational recommender systems (CRS) focus on eliciting user preference through strategically asking questions [5–8]. With the progress in conversational methods, this is becoming increasingly popular to obtain dynamic preference and recommend items [9, 10]. CRS can be thought of the bridge between search and recommendation. In a search problem, the user uses query to indicate their intention. Whereas in a recommendation problem, the RS agent tries to infer the user's intention through historical data. Multi-turn CRS is an interactive retrieval method that utilizes multiple question-answers and recommendations to infer the user's intention.

Christakopoulou et al. [11] proposed the idea of CRS, where they used multi-armed bandit setup to recommend items. The current works in CRS also mostly rely on reinforcement learning based policy learning methods [12–14]. Sun et al. [15] considered the problem of when to recommend in addition to which attributes to ask. Recent works in CRS use the setup proposed by Lei et al. [12]. In this setting, CRS can ask a question regarding an attribute of the items or recommend some items multiple times until the user accepts the recommendation or we reach the end of conversation. We use this setup in our study.

The current works in CRS are limited because the existing works rely on learning specific user's embedding to ask questions and make recommendations. This fails when a new user enters the system (the cold start problem). However, we believe that inherently CRS should be able to handle the cold-start problem in the following way. If a CRS model learns a generalized

policy, it can ask the right questions to elicit the current preference of any user, and thus it can create a profile of the new user. Therefore, the CRS should be able to recommend the target item to that user. Even though it is an easy problem to understand, it is not an easy problem to solve—because the preferences of different users may be very different, in which case, the same policy may not be sufficient to elicit all of their preferences. To the best of our knowledge, this problem has not been addressed nor studied in the current works in CRS. Additionally, we note that current CRS works mainly use reinforcement policy learning. We argue that the multi-round CRS setup can be handled using supervised learning.

In this thesis, we present our case on how we can overcome this problem by profiling new users using their interaction with different items. We do this by building a explainable supervised learning model. We proposed a way of learning user-interaction embeddings. Using the embeddings we learn a forest of decision trees, namely FACT-CRS (Factorization Tree based Conversational Recommender System), which we use to ask questions. If the user rejects our recommendations, our model can correct the user-interaction embedding. Using that user-interaction embedding and the learned set of item embeddings, our model is able to rank the top items to recommend.

In this chapter, we introduce conversational recommender system and identify two key limitations. In Section 1.1, we describe the problem that we were trying to solve in this thesis. The objective of our work and our contributions are discussed in Section 1.2 and 1.3 respectively. The organization of the thesis book is described in Section 1.4.

1.1 Problem Statement

The problem we are investigating can be informally stated as: how can we build a CRS using a supervised model that learns to profile new users using their online feedback and effectively recommend in a multi-round CRS? Here, effective recommendation refers to identifying the target item by asking the fewest number of questions and making the fewest number of recommendations.

The problem inherently raises the following research questions:

- **RQ1.** How do we decide which question to ask?
- **RQ2.** How do we decide when to recommend?
- **RQ3.** How do we improve when the user rejects recommended items?

1.2 Objective

There are two main objectives of any conversational recommender system:

Recall: We want to recommend items that the user is looking for. So the objective of the multi-round CRS is to maximize the number of successful conversations (or minimize the number of failed conversations). To do that, we need to ask the questions that quickly narrows down our candidate set of items.

Timeliness: We want to use as few inputs from the user as possible. For a specific conversation, each input from the user is called a turn. A turn can be either a question followed by a user's answer, or a Top-K recommendation which the user either accepts or rejects. If the user accepts the Top-K recommendation, we say the conversation is successful. The conversation automatically fails when we reach the end of the turn-limit.

In this study we are further interested in the following:

User-interaction-based rule induction. Our main objective is to learn the latent factors (embeddings) in any user's user-interactions and use the user-interaction embeddings for recommendation. Learning user specific embeddings fails when a new user enters the system. We believe if we can learn a good representation of user's individual user-interactions, we can infer the user profile by online interaction. Thus we can rank the items by using learned item embeddings and infered user-interaction embedding.

Sufficiency of supervised learning. The current approaches in CRS mostly rely on reinforcement policy learning methods. In this study, we demonstrate that multi-turn CRS can be modeled using supervised machine learning by using a decision tree based model.

1.3 Contributions

In this study, we propose a novel approach to think about conversational recommender systems by modeling user-item interaction using a decision tree, namely FACT-CRS. We use a rule-based decision making into the learning of latent factors for each user-item interaction. We use user-item interaction as the basis of the decision tree because we believe that different users can describe the same item using different attributes. Figure 1.2 shows an example of restaurant recommendation. It shows how we can build a model to ask questions in conversational recommender system leveraging the set of user-item interactions. In this figure, the user-item interaction is the attributes the user has used to describe that item. Item i_2 has been described by the user u_1 as having the attribute "Breakfast"—whereas u_2 describes the same item i_2 with no mention to "Breakfast". Additionally, u_2 mentions another attribute "Coffee" about i_2, which u_1 doesn't mention. That is why, instead of modeling using just users or just items, we model using

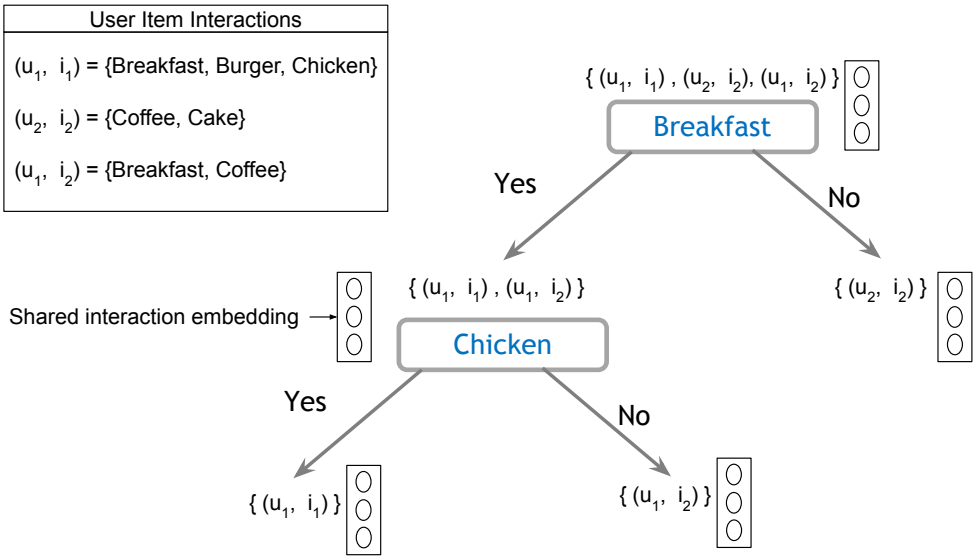

Figure 1.2: An example of user-item interaction tree in FACT-CRS. We can learn shared latent factors (embeddings) of user-item interaction using the interaction tree and use it to ask questions and recommend.

user-item interaction.

Next, let's look at the tree in Figure 1.2. The top level contains all the user-item interaction. If we split the interactions by "breakfast" we can see that item i_2 appears on both of the splits. This is exactly what we want: to learn all the ways different users describe an item. Also, if two interactions both mention an attribute "breakfast" in their reviews, they should be assigned to the same node on the tree to share the same latent factors (embeddings). Due to similar characteristics shared by each group of user-item interactions created by the learnt rules, the descriptive power of the learned shared-embedding is enhanced.

The main contributions of this study are as follows:

- We propose a CRS model using supervised learning which works for any user even if it has not previously interacted with that user. All the existing CRS methods [12,13,16,17] mainly rely on pre-trained user embeddings to make recommendations. But those information may not be available when a new user enters the system. In this study, we build a CRS model that does not rely on previously learned user's embeddings.

- Given a set of interactions, we provide an algorithm to learn the embeddings for each user-interactions. Each user-interaction in our model acts as a "super user". To the best of our knowledge, no prior work has studied learning the latent factors of user-interactions to rank the items in CRS. We further learn better representations of user-interaction and item embedding using user-interaction tree by using a Factorization Tree [18] method.

- We adapt FacT to meet the challenges in CRS, which are: 1) Which questions to ask 2)

When to recommend 3) How to handle user's feedback.

User-interaction Tree guides us on which questions to ask based on the user feedback. By asking those questions, we get an embedding for that user-interaction. We propose using multiple (a forest of) User-item Interaction. This gives us the ability to ask an arbitrary number of questions. We also propose a simple strategy to decide when to recommend items. This also allows us to make an early recommendation.

- We experimentally show that our supervised model, FACT-CRS, is sufficient to face the challenges in multi-turn conversational recommendations. FACT-CRS can efficiently ask questions and recommend items using two benchmark datasets, namely LastFM and Yelp. We further study the effectiveness of each component of our model. We also study the failed conversations and explain the reason behind failures. Experimental results show that FACT-CRS is able to outperform the reinforcement learning based models.

1.4 Organization of the Thesis

The rest of the book is organized as follows. In Chapter 2, we discuss the brief history of conversational recommendations and the existing works on CRS. The limitations of existing methods and the scope of our work are also discussed in this chapter. Our proposed approach, FACT-CRS, is introduced in Chapter 3. In this chapter, we discuss how we can learn user-interaction embedding and item embedding using user-interaction data. Chapter 4 discusses how we adapt the research questions to meet the challenges in CRS. We discuss the experimentation setups, dataset and user simulator in Chapter 5. Here, the experimental results are presented and compared against the baseline methods. Finally, Chapter 6 provides the concluding remarks and the scope of future works.

Chapter 2

Background and Related Works

We widely rely on recommender systems (RS) to seek information about new products, movies, songs, restaurants, shops etc. Recommender systems is one of the most studies topics in computer science, and they are found everywhere including restaurants, e-news, e-commerce, streaming services, social networks etc. While traditional recommenders predict the users' preferences from their historical interaction data, conversational recommender systems (CRS) leverage interactive conversations to dynamically elicit the users' preferences.

In this chapter, we discuss the inception of conversational recommender system. In Section 2.1, we briefly explain the limitations of traditional recommender systems. We discuss how CRS can overcome those limitations in Section 2.2. In Section 2.3, we describe the existing research directions in CRS. Section 2.4 outlines the existing challenges in CRS. In Section 2.6, we outline the scope of our work.

2.1 Limitations of Traditional Recommender Systems

Traditional recommender systems are static in the sense that they try to score the user's preference based on historical user-item interaction data and fail to elicit the uses' current preference. The traditional approaches can be broadly divided into two categories: 1) Content based and 2) Collaborative filtering.

2.1.1 Content-based Approaches

The content based approach assumes that for each item, the attributes associated with the item are known. In case of movies it can be the name of actors, directors, year of production etc. For news articles, the contents might be extracted using term-frequency inverse-document-frequence (TF-IDF). Based on that, the RS builds a model and suggests the items which are "most similar"

(e.g. dot product, cosine similarity etc.) to the items the user has interacted with. Content based approaches usually suffer from the following two issues:

Limited access: The RS may have limited access to information on the users and/or the contents. The users might be reluctant to share personal information because of privacy issues. Often precise content of items may be difficult to get for items which are not well-tagged, such as short video clips. Another key problem is that content of an item might not be sufficient to determine its quality. For articles, if we only use TF-IDF as the content profile, it might not be possible to differentiate between a well written and a poorly written article.

Lack of serendipity: Serendipity is a key feature of any RS. It refers to how well an RS can recommend new or different type of items to the users who would likely be interested those items. Here content based RS suffers greatly, because the RS will predict high rating for an item if similar items are already liked by the user. Likewise, if the user hasn't already interacted with an item of different type, the RS will predict lower rating for that item. It might be the case that the RS keeps suggesting a user the movies with the same actors or director that the user has previously watched.

2.1.2 Collaborative Filtering (CF)

Collaborative filtering approach makes some improvement on the shortcomings of content based recommendations. The key idea behind collaborative filtering is the preference of a user for a new item is similar to another user's preference of that new item if both of them rated other items similarly.

Neighborhood based

In this approach, RS calculates the similarity between users or items using the user-item rating matrix, and then recommends the items that are positively rated by similar users of the target user.

User Similarity: Usually, the similarity between any pair of users is calculated by comparing how they have rated the items. The popular similarity measures are cosine similarity and Pearson correlation coefficient (PCC).

Item Similarity: The item similarity based RS calculates the similarity between items according to the user-item rating matrix and groups the similar items. It then predicts unknown ratings on the target item using the neighbors.

Model based

In model-based RS, there is a training stage to learn the model parameters. Once the model is trained, the user-item rating prediction is very fast since it only needs the model parameters.

Matrix Factorization (MF): In this approach, the rating matrix $\mathbf{R}_{m\times n}$ is used to learn the low rank (let's say d dimensional) latent embedding of the users $\mathbf{U}_{m\times d}$ and latent embedding of the items $\mathbf{V}_{n\times d}$ according to some predefined objective function (e.g. L-2 loss), where m is the number of users and n is the number of items. $\mathbf{U}\mathbf{V}^T$ predicts the rating of all user item pairs.

Singular Value Decomposition (SVD): This is a special MF approach where the rating matrix $\mathbf{R}_{m\times n}$ is decomposed into $\mathbf{R}_{m\times n} = \mathbf{U}_{m\times m}\mathbf{S}_{m\times n}\mathbf{V}^T_{n\times n}$. Here, $\mathbf{U}\mathbf{U}^T = \mathbf{I}_{m\times m}$ and $\mathbf{V}\mathbf{V}^T = \mathbf{I}_{n\times n}$. Usually CF approach performs better since the predicted rating is based on the quality of items as evaluated by similar users. Also, it can recommend items with different type of contents.

2.1.3 Deep learning techniques

Deep neural architectures have been shown to be very successful in RS. Such techniques include Neural Collaborative Filtering, Convolutional Neural Network, Recurrent Neural Network, and Deep Reinforcement Learning based approaches.

Neural Collaborative Filtering

It is a feed-forward neural network with multiple hidden layers and non-linear activation functions. Given the user attributes x_u^{user} and the item attributes x_i^{item}, a multi-layer perceptron (MLP) f learns the parameter θ to predict the rating as :

$$\hat{r}_{u,i} = f(x_u^{user}, x_i^{item}|\mathbf{U}, \mathbf{V}, \theta)$$

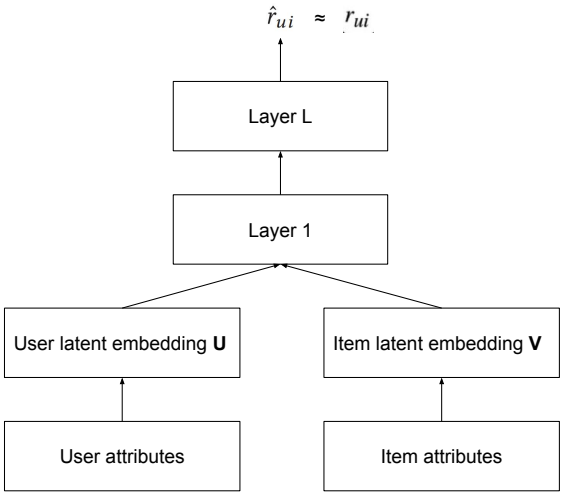

Figure 2.1: General structure of neural collaborative filtering

Convolutional Neural Network (CNN) based

CNN is a special neural network where in addition to linear layers and activation layers, there are convolution and pooling layers. CNN can learn the spatial representations in both global and

local settings. Recent works [19] have shown the effectiveness of visual description in restaurant recommendation. The representation of CNN in addition to textual representation are fed to other models to make recommendations.

Recurrent Neural Network (RNN)

RNNs are very effective to learn from sequential data. RNNs have internal parameters with feedback loop which are fed into the model in addition to the temporal data. There are many models which belong to the RNN family, such as Long Short Term Memory (LSTM), Gated Recurrent Unit (GRU) etc.

Hybrid Models

Hybrid models combine several deep learning approaches to learn better user and item representations. For example, Zhang et al. [20] utilizes CNNs and autoencoder to learn from visual data in collaborative knowledge (CKE) based embedding. CKE uses different moduls to learn from structural, textual, and visual data.

Even though many recent traditional approaches have been successfully deployed in a number of real world cases [2–4], they have some fundamental limitations:

- Traditional recommender systems do not take into account current preference of the user. For example: a user who usually likes a particular type of food (e.g. Mexican food) may want to try another type of food (e.g. Thai food). In this case, a traditional RS will not be able to infer because it does not have the access to user's current preference.
- The user might be looking for completely different type of item as illustrated in figure 2.2.
- The user's preference can change over time.
- It is difficult to learn the exact reason of why a user interacts with certain items since the traditional systems have no access to such information.

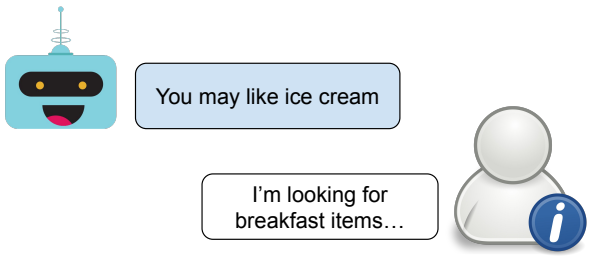

Figure 2.2: Limitation of traditional recommender systems

2.2 Alternative Approach: Conversational Recommender Systems

Conversational recommender systems (CRS) overcomes these limitations by eliciting user preference through dynamic real time interaction [5–8]. This area of research has gained a lot of attention since it can be used to obtain a user's current preference in order to recommend items [9, 10]. A CRS infers the dynamic preferences of users from the conversations, where the user tries to describe the target item by answering a multi-turn question answer game.

The definition of CRS can be very broad. A CRS retrieves information using mixed-initiative dialog between a user and agent. The agent's actions are optimized based on the current user needs in the current conversation, using both online and historical information of the user. Radlinski et al. [7] describes the following 5 properties of CRS known as the RRIMS property:

- User **R**evealment: The system helps the users recognize their need.
- System **R**evealment: The system informs the user its abilities and the scope of the recommendation.
- Mixed **I**nitiative: Both the user or the system may intiate the conversation [8].
- **M**emory: The user can reference past statements, which implicitly remain true unless contradicted.
- **S**et Retrieval: Both positive and negative feedback should narrow down the candidate set of items.

2.3 Types of Research in CRS

Early forms of CRSs can be traced to interactive recommender systems [21–23] and critiquing-based recommender systems [24–26]. They focus on improving the recommendation strategy online by leveraging real-time user feedback on previously recommended items. While surveying the literature, we found the following four main directions of current research in CRS:

2.3.1 Item based

In this direction of research, the system only asks whether a user likes items. The system does not ask attributes questions. This was the beginning of CRS when Christakopoulou et al. [11] attempted this line of research. They employed multi-armed bandit models to acquire the

users' feedback on individual items. In this setting, the items are the arms of the bandits, the environment is dependent on the user and the reward is the user's feedback (whether or not the user accepts that item). The model updates its parameters at each turn. For new users, the user embedding is initialized as the mean embedding of existing users.

2.3.2 Question Driven

In this setting, the domain was expanded so that the system needed to predict two things: 1) what questions to ask and 2) which items to recommend. Note that, in this setting the system only recommends once. So, the system didn't have to choose between question and recommendation i.e., "when" to ask and "when" to recommend. Zhang et al. [6] proposed a model which consisted of three stages. In the initiation stage, user initiates a conversation. In the conversation stage, the system asks the user preferences on attributes of items. In the final stage, the system recommends the items. Zou et al. [27] proposed Qrec which asks questions a predetermined number of times and then makes recommendation once. In Qrec, they chose the most uncertain attribute to ask (the attribute that the system has the smallest confidence between positive and negative feedback). Christakopoulou et al. [28] later proposed a method which facilitates multiple answers (for example: "choose all the attributes that you like.").

2.3.3 Multi-turn CRS

In this setting, another aspect was added to CRS. Here, a conversational recommender system needs to choose between asking questions and making recommendations in a multi-turn conversation. Sun et al. [15] used a belief tracker using an LSTM model to determine when to recommend, but their model was not able to handle when user rejected a recommendation. Lei et al. [12, 17] expanded the single-round CRS to the multi-round setting, where multiple questions and recommendations can be made in one conversation until the user accepts the recommendation or until the end of the conversation. They utilized three different modules: the estimation module estimates user preference on items and attributes; the action module decides whether to ask attributes or recommend items; and the reflection module updates the model when there is negative feedback. A dynamic preference modeling was proposed by Xu et al. [16]. They proposed a gating mechanism to include both positive and negative user feedback. Deng et al. [13] combined the question selection and recommender modules. They proposed two heuristics for reducing the candidate action space by pre-selecting attributes and items in each turn to simplify the RL training. Zhang et al. [29] used a bandit algorithm to select attributes and used a heuristic to decide whether to ask questions about attributes or make recommendations. Li et al. [30] unified attributes and items in the same arm space in a multi-armed bandit setting to determine the questions and used another bandit to determine when to recommend. Li et

al. [31] used a deep RL based bandit model to decide when to make a recommendation or which question to ask.

2.3.4 Dialogue based

There is another aspect of CRS, which is based on understanding and generating dialogs in natural language. This adds personalized response to the conversation and it is applicable to an even broader scope. Usually, in this approach, there is a natural language understanding module, and a natural language generation module. There is another module that keeps track of the state (current conversation). A dialog manager uses the dialog state to query from a knowledge base and sends it to another module which then learns the dialog policy. The dialogs in a session are generated sequentially, so it is formulated as a Markov Decision Process (MDP). Some of the current works use natural language processing in CRS [31–34]. This approach is beyond the scope of our work.

2.4 Challenges in Multi-turn CRS

In this study, we employ the multi-turn CRS approach. We break down the main challenges in multi-turn CRS as follows:

Decision. A main challenge in CRS is deciding whether to ask more questions or make a recommendation. Whereas asking more questions can potentially narrow down a candidate item size, making a recommendation can help lower number of user inputs in a conversation.

Questions. The key advantage of conversational recommendation is being able to ask questions. The system can ask about attributes of items to narrow down the recommended candidates.

Recommendations. Another challenge is choosing a proper strategy to rank the items when recommending.

Online Negative feedback. In multi-turn CRS, the recommender system gets multiple chances to get the target item. Therefore, when a user rejects a recommendation made by the agent, the agent should be able to identify what it did wrong, and correct itself for future.

2.5 Limitations of Current Approaches in CRS:

We identify the following limitations in the current approaches in CRS.

Reinforcement Learning based. The current approaches in CRS are all based on reinforcement learning agents. While reinforcement learning is very powerful in solving some problems that

are not solvable by conventional techniques, reinforcement learning models require a lot of training and computation. If we can model the problem in rule-based supervised setting, we can effectively reduce the training time.

Cold Start. All the existing [12, 13, 16, 17, 31] CRS methods mainly rely on pre-trained user embeddings to make recommendations. But those information may not be available for new users. For cold start user, the model fails to ask personalized questions, which leads to ineffective recommendations. In this study, we aim to build a CRS model that does not rely on previously learned user's embeddings. Rather it can learn a profile based on the current user-item interaction.

2.6 Scope of our work: Multi-turn Q&R CRS

In this chapter we presented various existing methods for CRS. In this thesis, we consider multi-round Questions and Recommendations (Q&R) setting in CRS with the following properties:

- We use templates to ask question. Binary interactions are asked in the format "Do you like attribute f_l?" (For example: "do you like coffee?")

  Multi-valued attributes are asked in the format: "What kind of value do you like about attribute f_l?" (For example: "what kind of coffee do you like?" The answer could be cappuccino, latte etc.)

- We consider multi-turn CRS, where at each turn, the system needs to make a decision between asking questions and making recommendations (Q&R).

- The agent will get multiple chances to recommend if rejected. This can continue until the user quits (i.e., maximum turn limit is reached).

- The system initiates the conversation.

- In a question turn, the user responds "Yes" to an attribute if and only if it is mentioned in that observed review, otherwise the user will respond "No".

Chapter 3

Methodology: FACT-CRS

In this chapter, we discuss in details the User-item Interaction tree based CRS approach used in our thesis. We explain our approach for user-item interaction latent factor (embedding) learning. One of the key contribution of our work is that we give much importance to user-item interactions because we belive it is the key to understanding the user's current preference. To the best of our knowledge, our work is the first that uses low-rank representation of user-item interactions to make recommendations. We learn the latent factors for user-item interactions as a function of rules. The intuition behind this is: the user-item interactions that provide the same responses to the same set of questions should share the same embeddings. The questions are selected from the textual attributes extracted from the user-item interactions. The questions are constructed recursively based on the previous subset of user-item interactions. Since the user's review of an item is the gateway to user's current preference, we use reviews as a proxy for the user-item interactions. However, we note that our model can handle user-item interaction in many forms such as clicks, ratings etc.

We construct a user-item interaction tree and and learn the embedding using FacT [18]. Each node in the tree has a predicate that represent the question to ask. Each node in the decision tree represents the members (set of user-item interactions). For simplicity, we first describe the model in terms of binary interaction (i.e., user either likes an attribute or does not), our model can easily adapt when the attributes are multi-valued, which is described in Section 3.4.

The preprocessing required for our model is described in section 3.2. This section discusses how we learn the initial user-item interaction and item embedding using matrix factorization. In Section 3.3 we discuss how we can jointly learn the user-item interaction embeddings and the item embeddings. Section 3.4 discusses the construction of user-item interaction tree. This tree is the main structure that we later use to ask questions.

3.1 Preliminaries

We denote the set of items as $\mathcal{I} = \{i_1, i_2, \ldots, i_n\}$ and the set of users as $\mathcal{U} = \{u_1, u_2, \ldots, u_m\}$. The set of attributes or features that are used to describe the items is $\mathcal{F} = \{f_1, f_2, \ldots, f_q\}$.

User-item Interactions. Suppose, user u has interacted with item i. We express this as a pair (u, i). User-item interaction denotes how user u has interacted with item i. This can be in terms of ratings, reviews, clicks etc. An example could be: user u describes item i in terms of $\{f_1, f_2\}$.

We can consider all the user-item interactions as the matrix $\mathbf{R} \in \{0, 1\}^{p \times q}$, where p is the number of interactions. Each user-item interaction r is a q dimensional vector ($\in \{0, 1\}^q$). Here, 0 means that the attribute is present in the review and 1 means that the attribute is not mentioned in the review.

Each CRS round constitutes of multiple turns. Each round can either be-

- QA: A question asked by the agent and followed by user's answer
- Top-K recommendations: When the agent decides to recommend (RQ2) we select the Top-K items that are ranked highest by our model and recommend those Top-K items to the user. This is followed by user's feedback. The user will either accept or reject the recommendation.

The questions follow this general pattern: "Do you like attribute f_j?" where $j \in \{1, 2, \ldots, q\}$, In our setting the possible user answers can be $\in \mathcal{A}$. We first describe the model in terms of binary interaction but we can easily extend the same idea when $\mathcal{A}$ is multi-valued, which is described in Section 3.4.

Table 3.1 summarizes the notations used throughout the thesis.

3.2 Initial Latent Factors Learning: User-item Interaction and Item embeddings

We use latent factor models to learn the representation of user-item interaction. Latent factor models have shown to perform well in modern recommender systems. Koren et al. [35] employed matrix factorization techniques which are further popularized by Factorization Machines [36]. This idea has been widely adopted by modern recommender systems. The main idea here is to find a low-rank vector representation (embeddings) of users and items. We utilize this method because it can learn the interactions and similarities between users and items (usually in the same vector space).

Table 3.1: Summary of notations

Symbol	Description
n	Number of items
m	Number of users
q	Number of attributes
p	Number of user-item interactions
$\mathcal{I}$	Set of items
$\mathcal{U}$	Set of users
$\mathcal{F}$	Set of attributes
$\mathbf{R}$	$\in \{0,1\}^{p\times q}$, User-item interaction matrix
$\mathbf{r}_j$	$\in \{0,1\}^{q}$, j^{th} user-item interaction
d	Dimension of latent factor representation (embedding) of user-item interaction
$\mathbf{s}_j$	$\in \mathbb{R}^d$, User-item interaction embedding of r_j
$\mathbf{v}_i$	$\in \mathbb{R}^d$, Item embedding of i
$\mathbf{S}$	$\in \mathbb{R}^{q\times d}$, Embedding matrix of all user-item interactions
$\mathbf{V}$	$\in \mathbb{R}^{n\times d}$, Embedding matrix of all items

In FACT-CRS, we use the idea of latent representation using decision tree based model [18]. FACT-CRS is very robust in the sense that it doesn't require a particular latent factor representation learning method. We use Matrix Factorization (MF) because it is simple, and easy to understand and implement. The initial latent factors are only used as a starting point, which are further refined and indexed using User-item Interaction Tree and Item Tree.

We have also experimented with different methods for learning embeddings. Finding good initial ("seed") latent factors can help FACT-CRS converge faster in the optimization step. Section 3.4 and 3.4.2 describes the process of getting the initial item embeddings.

3.2.1 Seed of User-item Interaction embedding

We consider the user-item interactions as the matrix $\mathbf{R} \in \{0,1\}^{p\times q}$. Here p is the number of user-item interactions. Each user-item interaction $\mathbf{r}$ is ($\in \{0,1\}^{q}$) where q is the number of attributes.

$$r_{j,l} = \begin{cases} 1, \text{ if attribute } f_l \text{ is mentioned in user-item interaction } \mathbf{r}_j \\ 0, \text{ otherwise} \end{cases} \tag{3.1}$$

The objective of latent factor is to learn the low rank embedding of each user-item interaction and each item in d dimensional vector space. The embedding of user-item interaction $\mathbf{r}_j$ is $\mathbf{s}_j$. $\mathbf{S} \in \mathbb{R}^{p \times d}$ is the matrix containing all user-item interaction embeddings.

Decomposing the user-item interaction matrix $\mathbf{R}$

User-item Interaction embeddings $\mathbf{S} \in \mathbb{R}^{p \times d}$ can be learnt by decomposing the user-item interaction matrix as:

$$\mathbf{R} = \mathbf{S}{\mathbf{F}_r}^T \tag{3.2}$$

Here, $\mathbf{F}_r \in \mathbb{R}^{q \times d}$ is the matrix that corresponds to all feature embeddings obtained using user-item interaction matrix decomposition, and $\mathbf{f}_l^r \in \mathbb{R}^d$ is feature embedding for feature f_l.

Objective

The decomposition in Equation 3.2 is done by minimizing the prediction error over a set of observed User-item Interactions as follows:

$$\mathcal{L}_R(\mathbf{S}, \mathbf{F}_r, \mathbf{R}) = \sum_{(j,l) \in \mathbf{R}} (r_{j,l} - \mathbf{s}_j^T \mathbf{f}_l^r)^2 \tag{3.3}$$

The User-item Interaction embeddings can be learnt by solving the following optimization problem:

$$(\mathbf{S}_0, \mathbf{F}_{r0}) = \arg\min_{\mathbf{S}, \mathbf{F}_r} \mathcal{L}_R(\mathbf{S}, \mathbf{F}_r, \mathbf{R}) + \lambda_{s_0} ||\mathbf{S}||_2 + \lambda_{f_{r_0}} ||\mathbf{F}_r||_2 \tag{3.4}$$

Here, λ_{s_0} and $\lambda_{f_{r_0}}$ are the corresponding coefficients of $||\mathbf{S}||_2$ and $||\mathbf{F}_r||_2$ in the L2 regularizer terms.

3.2.2 Seed of Item Embedding

We learn the seed of the item embedding by first defining the Item Opinion Profile Matrix $\mathbf{O}$ and then using matrix factorization.

Normalized Item Opinion Profile $\mathbf{O}^n$

Suppose, feature f_l is mentioned $p_{i,l}$ number of times in all user-generated user-item interactions about item i. We can construct a feature-level profile $\mathbf{O}^n$ for each item i, where each element $o_{i,l}^n$ is defined as:

$$o_{i,l}^{n} = \begin{cases} 0, & \text{if } \sum_{x=1}^{q} p_{i,x} = 0 \\ \frac{p_{i,l}}{\sum_{x=1}^{q} p_{i,x}}, & \text{otherwise} \end{cases} \tag{3.5}$$

Binary Item Opinion Profile $\mathbf{O}^b$

Similarly, we can construct a Binary Item Opinion Profile $\mathbf{O}^b$, where each element $o_{i,l}^b$ is defined as:

$$o_{i,l}^{b} = \begin{cases} 0, & \text{if } p_{i,l} = 0 \\ 1, & \text{if } p_{i,l} > 0 \end{cases} \tag{3.6}$$

Essentially $o_{i,l}^b$ refers to whether or not item i has been associated with the attribute f_l in any user generated user-item interaction in $\mathbf{R}$.

Decomposing the Item Opinion Profile matrix $\mathbf{R}$

Similar to user-item interaction embeddings, item embeddings $\mathbf{V} \in \mathbb{R}^{n \times d}$ can be learnt by decomposing the Item Opinion Profile matrix as:

$$\mathbf{O} = \mathbf{V}{\mathbf{F}_o}^T \tag{3.7}$$

Here, $\mathbf{F}_o \in \mathbb{R}^{q \times d}$ is the matrix that corresponds to all feature embeddings obtained using item opinion profile matrix decomposition, and $\mathbf{f}_l^o \in \mathbb{R}^d$ is individual feature embedding.

Objective

The decomposition in Equation 3.7 is done by minimizing the prediction error over the item opinion profile as follows:

$$\mathcal{L}_O(\mathbf{V}, \mathbf{F}_o, \mathbf{O}) = \sum_{(i,l) \in \mathbf{O}} (o_{i,l} - \mathbf{v}_i^T \mathbf{f}_l^o)^2 \tag{3.8}$$

The item embeddings can be learnt by solving the following optimization problem:

$$(\mathbf{V}_0, \mathbf{F}_{o0}) = \arg\min_{\mathbf{V}, \mathbf{F}_o} \mathcal{L}_I(\mathbf{V}, \mathbf{F}_o, \mathbf{O}) + \lambda_{v_0} ||\mathbf{V}||_2 + \lambda_{f_{o_0}} ||\mathbf{F}_o||_2 \tag{3.9}$$

Here, $||\mathbf{V}||_2$ and $||\mathbf{F}_o||_2$ are the L2 norms, and λ_{v_0} and $\lambda_{f_{o_0}}$ are the corresponding coefficients to reduce the model complexity.

3.3 Joint optimization of User-item Interaction and Item embedding

The seed user-item interaction embeddings $\mathbf{S}_0$ and item embeddings $\mathbf{V}_0$ learnt from the previous section 3.2 are now used to learn better user-item interaction and item embeddings using Binary User Interaction Matrix $\mathbf{B}$ and Stochastic Gradient Descent (SGD).

3.3.1 Binary User Interaction Matrix B

We obtain the Binary User Interaction matrix $\mathbf{B} \in \{0,1\}^{m \times n}$ which represents whether or not the user has interacted with an item. Each element $b_{i,j}$ is defined as:

$$b_{i,j} = \begin{cases} 1, & \text{if user } u \text{ has interacted with item } i \text{ in } \mathbf{R} \\ 0, & \text{otherwise} \end{cases} \tag{3.10}$$

3.3.2 Joint Optimization

Suppose, $\mathbf{r}_j$ is user u's description of interaction with item i. $\mathbf{s}_j$ is the user-item interaction embedding of $\mathbf{r}_j$ and $\mathbf{v}_i$ is the item embedding of i. Now, we want to refine our learned $\mathbf{s}_j$ and $\mathbf{v}_i$. We can do this by minimizing the prediction error over a set of observed user-item interaction. First, we define the Cross-Entropy (CE) loss as:

$$\mathcal{L}_{CE}(\mathbf{S}, \mathbf{V}, \mathbf{B}) = \sum_{r_j \in \mathbf{R}} (b_{u,i} - \sigma(\mathbf{s}_j^T \mathbf{v}_i)) \tag{3.11}$$

Where, $\sigma(\cdot)$ is the Sigmoid or Logistic function:

$$\sigma(x) = \frac{1}{1 + \exp(-x)}$$

Bayesian Pairwise Ranking (BPR) Loss via Negative Item Sampling

The loss function in Equation 3.11 is not sufficient in the sense that it cannot fully capture the relative ranking of the items [37, 38]. In recommender systems in general, we care about the relative ranking of the items. Bayesian Personalized Ranking (BPR) loss [37] has been very widely used in latent factor representation learning to better learn the relative item ranking.

We can employ the BPR loss by creating a subset of negative items for user u using the following steps: Let $\mathbf{r}_j$ be the user-item interaction of (u, i) and let $\mathbf{s}_j$ be the user-item interaction embedding of $\mathbf{r}_j$. We create a set of negative samples D_j^{neg} for $\mathbf{r}_j$ by randomly choosing N_{BPR}

items i_{neg} where $b_{u,i_{neg}} = 0$. Now, the BPR Loss is calculated as:

$$\mathcal{B}\left(\mathbf{s}_j, \mathbf{V}, D_j^{neg}\right) = \frac{1}{N_{BPR}} \sum_{i_{neg} \in D_j^{neg}} \log \sigma\left(\mathbf{s}_j^T \mathbf{v}_i - \mathbf{s}_j^T \mathbf{v}_{i_{neg}}\right) \tag{3.12}$$

The user-item interaction embeddings and item embeddings can be jointly learnt by solving the following optimization problem using Stochastic Gradient Descent (SGD):

$$(\hat{\mathbf{S}}, \hat{\mathbf{V}}) = \arg\min_{\mathbf{S},\mathbf{V}} \mathcal{L}_{CE}(\mathbf{S}, \mathbf{V}, \mathbf{B}) + \lambda_{BPR} \sum_j \mathcal{B}\left(\mathbf{s}_j, \mathbf{V}, D_j^{neg}\right) + \lambda_s ||\mathbf{S}||_2 + \lambda_v ||\mathbf{V}||_2 \tag{3.13}$$

Here, $||\mathbf{S}||_2$ and $||\mathbf{V}||_2$ are the L2 norms and λ_{BPR}, λ_v and λ_s are the corresponding coefficients in the regularization terms.

Algorithm 1 summarizes the steps to get the initial user-item interaction and item embeddings.

Algorithm 1 Steps to get the initial item and user-item interaction embeddings

1: **procedure** GET INITIAL REVIEW AND ITEM EMBEDDING($\mathbf{R}, \mathbf{O}, \mathbf{B}$)
2: Seed $\mathbf{S}_0, \mathbf{F}_{r0} = \arg\min_{\mathbf{S},\mathbf{F}_r} \mathcal{L}_R(\mathbf{S}, \mathbf{F}_r, \mathbf{R}) + \lambda_{s_0}||\mathbf{S}||_2 + \lambda_{f_{r0}}||\mathbf{F}_r||_2$
3: Seed $\mathbf{V}_0, \mathbf{F}_{o0} = \arg\min_{\mathbf{V},\mathbf{F}_o} \mathcal{L}_O(\mathbf{V}, \mathbf{F}_o, \mathbf{O}) + \lambda_{v_0}||\mathbf{V}||_2 + \lambda_{f_{o0}}||\mathbf{F}_o||_2$
4: $(\hat{\mathbf{S}}, \hat{\mathbf{V}}) = \arg\min_{\mathbf{S},\mathbf{V}} \mathcal{L}_{MSE}(\mathbf{S}, \mathbf{V}, \mathbf{B}) + \lambda_{BPR} \sum_j \mathcal{B}\left(\mathbf{s}_j, \mathbf{V}, D_j^{neg}\right) + \lambda_s||\mathbf{S}||_2 + \lambda_v||\mathbf{V}||_2$
return $\hat{\mathbf{S}}, \hat{\mathbf{V}}$
5: **end procedure**

After getting the initial embeddings, we are now ready to create the user-item interaction tree.

3.4 User-item Interaction Tree

Based on the user-item interaction (u, i) content $\mathbf{r}_j$, we can check whether or not a specific attribute f_l is mentioned. We can do this by checking wheter $r_{j,l} = 0$. This allows us to hierarchically model the latent factors as a function of the attributes $\mathcal{F}$. For each attribute f_l we can divide the subset of user-item interactions $\mathbf{R}_a \subseteq \mathbf{R}$ into 2 disjoint sets Pos and Neg as given in Equation 3.14. This is illustrated in Figure 3.1.

$$\begin{aligned} Pos\left(f_l \mid \mathbf{R}_a\right) &= \{\mathbf{r}_j \mid r_{j,l} = 1, \mathbf{r}_j \in \mathbf{R}_a\} \\ Neg\left(f_l \mid \mathbf{R}_a\right) &= \{\mathbf{r}_j \mid r_{j,l} = 0, \mathbf{r}_j \in \mathbf{R}_a\} \end{aligned} \tag{3.14}$$

Optimal Partition

The next question is: given the subset of user-item interactions $\mathbf{R}_a$, how can we select the best

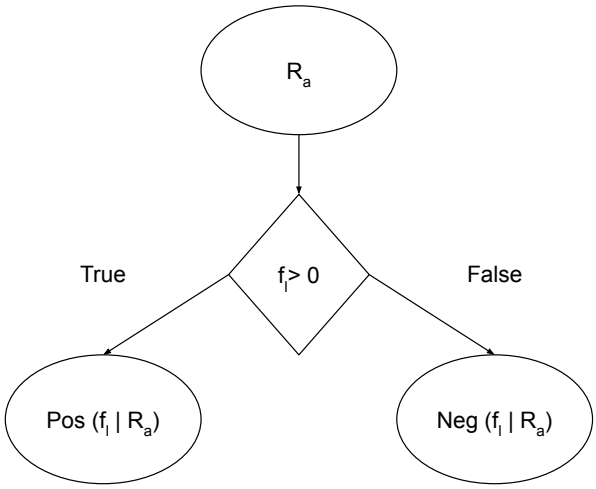

Figure 3.1: The subset of user-item interactions R_a is split into two subsets based on the attribute f_l

attribute f_l? An optimal attribute split would partition the user-item interaction subset $\mathbf{R}_a$, where the latent factors in each disjoint group minimizes SGD Loss as given by Equation 3.11. We can find the optimal attribute f_l^* by exhaustively searching through the attribute set $\mathcal{F}$. Let, $\mathbf{R}_{pos}$ and $\mathbf{R}_{neg}$ be the observed user-item interactions in the resulting 2 partitions, and $\mathbf{s}_{pos}, \mathbf{s}_{neg}$ be the corresponding user-item interaction embeddings in each of the partitions. We can find the optimal split using Equation 3.15 and Equation 3.16.

$$\begin{aligned}\mathcal{L}(f_l) = \min_{\mathbf{s}_{pos},\mathbf{s}_{neg},\mathbf{V}} \{ & \mathcal{L}_{CE}(\mathbf{s}_{pos}, \mathbf{V}, Pos\left(f_l \mid \mathbf{R}_a\right)) + \lambda_{BPR} \sum_{\mathbf{s}_j \in Pos(f_l|\mathbf{R}_a)} \mathcal{B}\left(\mathbf{s}_j, \mathbf{V}, D_j^{neg}\right) + \lambda_s ||\mathbf{s}_{pos}||_2 \\ & + \mathcal{L}_{CE}(\mathbf{s}_{neg}, \mathbf{V}, Neg\left(f_l \mid \mathbf{R}_a\right)) + \lambda_{BPR} \sum_{\mathbf{s}_j \in Neg(f_l|\mathbf{R}_a)} \mathcal{B}\left(\mathbf{s}_j, \mathbf{V}, D_j^{neg}\right) + \lambda_s ||\mathbf{s}_{neg}||_2 \}\end{aligned} \tag{3.15}$$

$$f^* = \arg\min_{f_l \in \mathcal{F}} \mathcal{L}(f_l) \tag{3.16}$$

Note that $\mathbf{s}_{pos}$ is the shared embedding of all the user-item interactions $\in Pos\left(f_l \mid \mathbf{R}_a\right)$ and $\mathbf{s}_{neg}$ is the shared embedding of all the user-item interactions $\in Neg\left(f_l \mid \mathbf{R}_a\right)$.

Building the User-item Interaction Tree

For each selected attribute f_l, equation 3.15 uses matrix factorization to learn the latent factors on the resulting disjoint sets to minimize the joint optimization loss $\mathcal{L}_{CE}$. However, we note that our model can use other latent factors models using any model of our choice. It is completely independent of the tree construction.

Starting with the root node where $\mathbf{R}_a = \mathbf{R}$, our attribute splitting (Equation 3.15) can be recur-

sively applied on the resulting user-item interaction partitions $Pos\left(f_l \mid \mathbf{R}_a\right)$ and $Neg\left(f_l \mid \mathbf{R}_a\right)$. This is how we get a Decision Tree like structure, where each node contains a subset of user-item interactions and corresponding learned user-item interaction embeddings. We refer to this as the user-item interaction tree. In general, each node is a structure $\{\text{member}, \text{embedding}\}$. For example, for positive split, member is the subset of user-item interactions $Pos\left(f_l \mid \mathbf{R}_a\right)$ and embedding is $\mathbf{s}_{pos}$. We terminate the process when the either of the following occurs:

1. The user-item interactions $\mathbf{R}_a$ at the current node cannot be split any further. In this case, we have reached an individual user-item interaction.
2. The maximum tree depth H_{max} has been reached.

Personalization

By design, all the members (user-item interactions in this case) in the same node share the same embedding. So, in the leaf nodes, we do another level of optimization using Equation 3.11 to find individual user-item interaction embeddings.

3.4.1 Multi-valued Attributes

Although we described the user-item interaction tree for binary attributes i.e., $\mathcal{A} = \{0, 1\}$, we can easily extend the idea for multi-valued attributes. Suppose, for attribute f_l, we have the possible outcomes $\mathcal{A}_l = \{a_1, a_2, \ldots, a_{|\mathcal{A}_l|}\}$. In this setting, each possible value will create a new branch using the Equation 3.17.

$$Branch\left(f_l = a_x \mid \mathbf{R}_a\right) = \{\mathbf{r}_j \mid r_{j,l} = a_x, \mathbf{r}_j \in \mathbf{R}_a\} \tag{3.17}$$

3.4.2 Item Embedding

To create the item tree, we make use of the Item Opinion Profile $\mathbf{O}$. It is slightly different from from before, because based on the item opinion o_i about item i, we need to check a specific attribute f_l against a threshold t.

Handing Continuous Values

We can do this by checking $o_{i,l} > t_l$. For each attribute f_l we divide the subset of item opinions $\mathbf{O}_a \subseteq \mathbf{O}$ into 2 disjoint sets Pos and Neg as given in Equation 3.18.

$$\begin{aligned} Pos\left(f_l \mid \mathbf{O}_a\right) &= \{\mathbf{o}_i \mid o_{i,l} > t_l, \mathbf{o}_i \in \mathbf{O}_a\} \\ Neg\left(f_l \mid \mathbf{O}_a\right) &= \{\mathbf{o}_i \mid o_{i,l} \leq t_l, \mathbf{o}_i \in \mathbf{O}_a\} \end{aligned} \tag{3.18}$$

Similar to User-item Interaction Tree, an optimal attribute split would partition the item subset O_a, where the latent factors in each disjoint group minimizes SGD Loss as given by Equation 3.11. We can find the optimal attribute f_l^* by exhaustively searching through the attribute set $\mathcal{F}$. Let, $\mathbf{O}_{pos}$ and $\mathbf{O}_{neg}$ item profiles in the resulting 2 partitions, and $\mathbf{v}pos, \mathbf{v}_{neg}$ be the corresponding item embeddings in each of the partitions. We can find the optimal split using Equation 3.19.

$$f_l^*, t_l^* = \arg\min_{f_l \in \mathcal{F}} \{\mathcal{L}_{CE}(\mathbf{S}, \mathbf{v}_{pos}, Pos\left(f_l \mid \mathbf{O}_a\right)) + \mathcal{L}_{CE}(\mathbf{S}, \mathbf{v}_{neg}, Pos\left(f_l \mid \mathbf{O}_a\right)) + \lambda_s(||\mathbf{v}_{pos}||_2 + ||\mathbf{v}_{neg}||_2)\} \tag{3.19}$$

$\mathbf{v}_{pos}$ is the shared item embeddings of the positive partition based on $o_{i,l} > t$ and $\mathbf{v}_{neg}$ is the shared item embeddings of the positive partition. In this case, in addition to finding the best attribute split, an additional question arises: how can we select the optimal threshold t? There can be two cases:

Using $\mathbf{O}^b$. This is similar to user-item interaction tree creation and can be solved by setting $t = 0$.

Using $\mathbf{O}^n$. One way we can solve it is by exhaustively searching through the attribute set $\mathcal{F}$ and possible values of threshold t. Since, the values in $\mathbf{O}^n$ are normalized, the possible values of $t \in [0, 1]$, which is a continuous space. However, Zipf's law dictates that the rank is inversely proportional to frequency , which leads the item opinion profiles to be concentrated [39]. We use equal width Interval binning [40] as a simple discretization technique.

Alternate Optimization

The algorithm described in Section 3.4 and 3.4.2 to build the user-item interaction tree is recursive. It requires that we already have learnt the item embeddings. Similarly, building the item tree requires that we have already learned the user-item interaction embeddings. We alternately build user-item interaction tree and item tree by iteratively optimizing Equation 3.15 and 3.19.

At the beginning (t = 0), we start building the user-item interaction tree from the initial item embedding learned using Algorithm 1. We use this user-item interaction tree to learn the user-item interaction embeddings. In general, at iteration t, we build user-item interaction tree using item embeddings $\mathbf{V}_{t-1}$. We use SGD to optimize Equation 3.15. When the user-item interaction tree construction terminates, we use personalization to learn the updated item embedding $\mathbf{V}_t$. We then use $\mathbf{V}_t$ to construct the user-item interaction tree and learn $\mathbf{S}_{t+1}$ using Equation 3.19. We repeat this process in Equation 3.15 and 3.19 until either:

1. The difference of losses in two successive iterations is smaller than some predefined value ζ; or
2. The iterations limit is reached

At iteration t = 0, we could also start constructing item tree using the initial $\mathbf{S}$.

Parallelization

We can easily parallelize the step in Equation 3.15 for user-item interaction tree and Equation 3.19 for item tree. We can do this because the while finding the best attribute we need to go through all the attributes. But the attributes themselves do not have any inter-dependency, so they can be parallelized–which significantly reduces the training time.

In this alternate optimization approach, the internal nodes are discarded between one iteration to another. Essentially this process of tree building is a type of hierarchical clustering [41]. The internal nodes in the user-item interaction tree can learn similarity among the user-item interaction (user-item interactions) in the same cluster (node).

Embedding Refinement from Parent Factors (PF)

We use residual approach to learn the embeddings from parent node to the child node. Let, at an internal node, the user-item interaction embedding is $\mathbf{s}$. Now, for attribute f_l, we represent embedding of the positive and negative split respectively as $\mathbf{s}_{pos} = \tilde{\mathbf{s}}_{pos} + \mathbf{s}$ and $\mathbf{s}_{neg} = \tilde{\mathbf{s}}_{neg} + \mathbf{s}$ where $\tilde{\mathbf{s}}_{pos}$ and $\tilde{\mathbf{s}}_{neg}$ are learned using Equation 3.15. $\tilde{\mathbf{s}}_{pos}$ and $\tilde{\mathbf{s}}_{neg}$ can be thought as refinement or residual corrections to the parent's embedding. This provides another perspective to think about the personalization step mentioned above. The leaf nodes' embedding can be thought of individual member's embedding. Whereas each internal node's embedding captures the embedding of the shared attributes in those members.

Now that we have built the user-interaction Tree and have learned the item embeddings, we are ready to ask questions and recommend in the CRS setting.

Chapter 4

Adapting FACT-CRS to meet the challenges in CRS

In this Chapter, we describe how the model we introduced in Chapter 3 can adapt to meet the research questions at hand. We introduced three research questions in Section 1.1–which are adjusted based on our model in Section 4.1. Section 4.2, 4.3 and 4.4 discusses the three research questions and our approach to solving them.

4.1 Adjusting the research questions based on our model

We discussed the following research questions in Chapter 1.1, which we adjust based on the model and the needs here.

RQ1. Initially, the research question was "which question to ask?". Decision Tree gives us a strategy to ask questions while taking into account the user's feedback. However, given the maximum depth of the decision tree is fixed H_{max}, we cannot ask more than H_{max} number of questions. For our model to generalize, we need to be able to ask arbitrary number of questions. So, we redefine this RQ as: "Given the depth of the decision tree is fixed, how can we ask arbitrary number of questions"?

RQ2. Initially the research question was stated as "when to recommend?". Decision Tree gives us a way to decide that in the following way. When we reach the leaf node of the tree, we get the predicted user-item interaction embedding. We can use the predicted user-item interaction embedding and the learned item embeddings to rank the items and recommend the top-K items. This would mean that we need to ask at most H_{max} questions before we can start recommending. However, an important goal of conversational recommender systems is to minimize the number of interaction with the user. In this case, we need to adapt and check if we can make recommendations before reaching the leaf of the tree. So the research question is

redefined as: "How can we decide to make an early recommendation before reaching the leaf of the tree"?

RQ3. We discussed the problem of how to improve when the user rejects recommended items. This is a research question that the modern conversational recommender systems hugely suffer from and do not have a good way of handling. As we will see later in the experiments, most CRS models cannot do much better than to keep making more and more recommendations until the end of the turn is reached, and hope that the target item will appear somewhere in the recommendations. The question is, can we do something different and better to understand what the user is looking for when they reject a recommendation? So, we redefine the research question to "how do we retain the information about the rejected items in a single conversation using the decision tree"? We propose a strategy to learn from the rejected recommendations in Section 4.4.

4.2 RQ1: How to ask an arbitrary number of questions?: A Random Forest-based Approach

Our approach to tackling this challenge is to build a random forest of N pairs of user-item interaction tree and item embeddings. We can do it in parallel so that each of the user-item interaction trees in the forest considers a maximum of f_{max} attributes where $f_{max} \leq q$. These attributes are randomly sampled from the set $\mathcal{F}$. We keep the maximum depth of each tree fixed and set it to H_{max} for each of the user-item interaction tree.

Robustness of User-Item Interaction Tree

User-item Interaction Tree is very robust to the user's noisy response, as we explain here. First, let's look at how item based clustering in insufficient. In the item based clustering, the root node contains all the items in $\mathcal{I}$. The children of any node in the tree (Pos and Neg) are disjoint sets. As we ask questions, we end up taking one of the branches, either Pos or Neg. If we just used item tree, we couldn't handle the user's noisy response. For example: if the user doesn't mention an attribute that is actually present in the target item (perhaps because the user does not care about that attribute), we will end up taking a wrong branch i.e., the child node that doesn't have the target item.

Figure 4.1 shows this problem. Assume that in a conversation, user u is looking for target item i. The current node contains the set of items $\mathcal{I}_a$, where $i \in \mathcal{I}_a$. Suppose, the predicate in that node is $f_l > 0?$, and based on positive response we get a subset of items $\mathcal{I}_{af_l+} \subseteq \mathcal{I}_{af_l-}$. Similarly, based on negative response we get a subset of items $\mathcal{I}_{af_l-} \subseteq \mathcal{I}_a$. Here, $\mathcal{I}_{af_l+} \cap \mathcal{I}_{af_l-} = \emptyset$. We further assume that that in the item opinion profile $\mathbf{O}$, the target item i is associated with attribute f_l; in other words, $o_{i,l} > 0$. This means $i \in \mathcal{I}_{af_l+}$. Now, when we ask the user, "Do you like attribute f_l?", it is possible that the the user forgets to mention that attribute and mistakenly

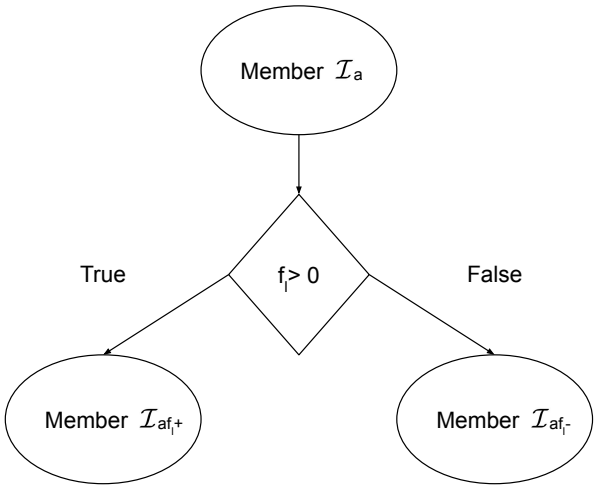

Figure 4.1: User-item Interaction tree solves the problem of noisy user response

answers "No". In that case, we will take the wrong branch and will end up with subset of items I_{af_l+}. However, $i \notin I_{af_l-}$, so we will completely miss that item.

Here, user-item interaction tree comes to the rescue. User-item interaction tree is more robust to this kind of noise because user-item interaction tree partitions based on user-item interaction content. Each interaction in a node the tree is actually an user-item interaction (u, i) in $\mathbf{R}$. So, when we partition the user-item interaction, the items on both partitions will not necessarily be disjoint.

Going back to the previous example, in the user-item interaction tree, consider (u, i) in a node with user-item interaction $\mathbf{R}_a$. Now consider only the item i in each pair (u, i) in the interactions $\mathbf{R}_a$, and denote that set of items as $\mathcal{I}'_a$. Now, assume that the target item $i_{target} \in \mathcal{I}'_a$. Assume, the predicate in that node is $f_l > 0?$, the positive branch contains the items $\mathcal{I}'_{af_l+} \subseteq \mathcal{I}'_a$ and the negative branch contains $\mathcal{I}'_{af_l-} \subseteq \mathcal{I}'_a$. However, in this case $\mathcal{I}'_b$ and $\mathcal{I}'_c$ are not necessarily disjoint, so it could be the case that $i_{target} \in \mathcal{I}'_{af_l+} \cap \mathcal{I}'_{af_l-}$, which means that we can still find the target item in spite of the noisy response.

Predicting Item Scores

Suppose, after traversing the user-item interaction tree the predicted embedding is $\mathbf{s}_{pred}$. We also learn the embedding of all items is $\mathbf{V}$. We score each item using:

$$Score(i) = \mathbf{s}_{pred}^T \mathbf{v}_i \tag{4.1}$$

Which User-item Interaction Tree First?

Now the question arises: out of all the trees in the forest, which one do we select first? Our strategy is to start with the tree that has the best cross validation success rate.

Next User-item Interaction Tree?

Next question is: assume that we have finished traversal of the first tree and made a recommendation. If the user rejects, how do we select the next tree? One way would be to randomly pick one tree. However, we believe that we can do better by using a strategy:

Closest Tree First (CTF): Suppose, we have asked/traversed a total of N_{asked} user-item interaction trees, and we have asked the set of features $\mathcal{F}_{asked} \subseteq \mathcal{F}$. Using the last tree, we predict the user-item interaction embedding $\mathbf{s}_{pred}$. We will try to traverse the remaining $N - N_{asked}$ user-item interaction trees in parallel using the attributes in $\mathcal{F}_{asked}$. Assume that we end up with the user-item interaction embeddings $\mathcal{S} = \{\mathbf{s}_j, \mathbf{s}_{j+1}, \ldots, \mathbf{s}_{pred_{j+N-N_{asked}-1}}, \}$. We can now rank the trees using the $s \in \mathcal{S}$ which is most similar to $\mathbf{s}_{pred}$. We use dot product as the similarity metric but any other similarity metric (e.g. Cosine similarity, Pearson's Correlation Coefficient etc.) can be used here. We give score to a User-item interaction Tree j as $\mathbf{s}_j^T \mathbf{s}_{pred}$. In Closest Tree First (CTF), we take the user-item interaction tree j with the highest similarity score.

Our experiments show that the mean rank of the target item is very close to K in the top-K recommendation. This means that we are usually very close to finding the correct item So, it is important that we retain information from the previous trees. Experimental results agree that CTF indeed performs better.

4.3 RQ2: Can we make early recommendations?

We have seen that we can make a recommendation when we reach the leaf node of the tree. However, sometimes it might be desirable to make prediction when we are "confident" about the item we want to recommend. We use the following two strategies to make early recommendations:

4.3.1 During Training: Prunning

After we build each pair of user-item interaction tree and item embedding, we prune the user-item interaction tree when the items in a node is "homogeneous", i.e., we have few different items in the node. We use Gini Index [42] for this purpose. Suppose, for an internal node in the user-item interaction tree, the set of items from the user-item interaction $\mathbf{R}_a$ is $\mathcal{I}'_a$. The predicate in that node is $f_l > 0?$. This partitions $\mathbf{R}_a$ into $\mathbf{R}_{af_l+}$ and $\mathbf{R}_{af_l-}$. Based on positive response we get a subset of items $\mathcal{I}'_{af_l+} \subseteq \mathcal{I}'_a$ and based on negative response we get a subset of items $\mathcal{I}'_{af_l-} \subseteq \mathcal{I}'_a$. We calculate Gini Index as:

$$\text{Gini Index } = 1 - \left(\frac{|\mathcal{I}_a|}{|\mathbf{R}_a|}\right)^2 \tag{4.2}$$

If the Gini Index of a node is greater than some predetermined threshold τ, we make that node a leaf node by pruning the children of that node.

4.3.2 During Testing: Top-K items

Following the works of [12, 13, 16] we use Top-K recommendation. So, while testing, if we encounter a node in the user-item interaction tree that has no more than K-items, we can make an early recommendation. As we have previously explained, each members of a node is an interaction (u, i) in $\mathbf{R}_a$. Let's say the set of items in that node is $\mathcal{I}_a$. Then, if $|\mathcal{I}'_a| \leq K$, we make an early recommendation. If strictly $|\mathcal{I}'_a| < K$, then we rank all then items $\in \mathcal{I} - \mathcal{I}'_a$ using the scoring function in Equation 4.1. The remaining $K - |\mathcal{I}'_a|$ items with the highest scores are included in the Top-K recommendation.

4.4 RQ3: How to handle negative feedback?

Negative feedback or recommendation rejection is an important challenge in CRS. When we make a recommendation and the user rejects, that gives us a signal that we can improve our approach. However, the current approaches in CRS cannot handle this negative feedback very well. We try to approach this problem by first identifying potential reason for failure and how to correct the learning during testing. For the discussion we split the RQ into two parts.

4.4.1 How to retain information from previously asked user-item interaction trees?

We only need to ask questions/traverse a new user-item interaction tree when the user rejects our recommendation. Cross validation in our experiments show that when we make recommendations, the mean rank of the target item is very close to K in the top-K recommendation. This means that we are usually very close to finding the correct item So, it is important that we retain information from the previous trees.

Suppose, we have asked/traversed a total of N_{asked} user-item interaction trees, and we have predicted the set of user-item interaction embeddings $\{s_{pred_1}, s_{pred_2}, \ldots, s_{pred_{N_{asked}}}\}$. Also, assume, the corresponding item embeddings are $\{\mathbf{V}_1, \mathbf{V}_2, \ldots, \mathbf{V}_{N_{asked}}\}$. We retain the information in the previous trees using the modified scoring function as:

$$Score(i) = \frac{1}{N_{asked}} \sum_{j=1}^{N_{asked}} \mathbf{s}_{pred_j}^T \mathbf{v}_{ji} \tag{4.3}$$

4.4.2 How can we make correction to the predicted user-item interaction embedding?

Let, interaction embedding from user-item interaction tree be $\mathbf{s_{pred}}$. As described in subsection 4.3.2, two disjoint set of items can be in the Top-K recommendation:

1. Items in the node of the user-item interaction tree $\mathcal{I}'_a$
2. Items with the highest scores not in the node of the user-item interaction tree (i.e., in $\mathcal{I} - \mathcal{I}'_a$). We denote this set of recommended items as $\mathcal{I}_R$

In our experiments, we have found that in successful recommendations, the target item mostly appears on $\mathcal{I}'_a$. That is why, if the user rejects our recommendation, we correct the predicted user-item interaction embedding to penalize the items in $\mathcal{I}_R$. So, the corrected user-item interaction embedding after rejection becomes:

$$\mathbf{s}'_{pred} = \mathbf{s}_{pred} - \alpha \frac{\sum_{i \in \mathcal{I}_R} \mathbf{v}_i}{|\mathcal{I}_R|} \tag{4.4}$$

Here, α is a hyper-parameter that determines how much we penalize the items in $\mathcal{I}_R$.

Chapter 5

Experiments

In this chapter, we discuss the experiments we have performed in this study. The dataset used in our experiments are described in Section 5.1. Section 5.2 describes the simulator used in our experiments. We discuss the state of the art baselines that we compare our model with in Section 5.3. Section 5.4 discusses the training steps of the user-item interaction trees and item embeddings. We also present the results of our experiments in a quantitative manner. In Section 5.5, we describe the metrics used in conversational recommendation. We compare the overall performance of our model with the baselines in Section 5.6. We use ablation study to find the importance of each of our components in Section 5.7. Section 5.9 summarizes our objectives, and the insights we have obtained through our experiments.

5.1 Datasets

We evaluated FACT-CRS on Yelp and LastFM datasets, which are two widely used benchmark datasets used in CRS [12, 13, 16]. We use the data curated by Lei et al. [12] for both datasets. They pruned the users with fewer than 10 reviews [2, 37] to reduce data sparsity.

LastFM [43] is a dataset for music artist recommendation. Lei et al. processed the original attributes by combining synonyms and removing low frequency attributes. They categorized the original attributes into 33 coarse-grained attributes. This dataset uses binary attributes, i.e., either an attribute is present or absent in the review.

Yelp is a business dataset for restaurant recommendation [44]. It contains restaurant reviews and user rating of the restaurants. Lei et al. [17] manually built two level taxonomy on the attributes. There are 29 first-level attributes such as “coffee and tea”, “event planning and services”, “dessert types” etc., with a total of 590 attributes. We used the first level attributes for our multi-turn setting. The statistics of the datasets are shown in Table 5.1.

Table 5.1: Summary of datasets

	LastFM	Yelp
# Users	1,801	27,675
# Items	7,432	70,311
# Interactions	76,693	1,368,606
# Attributes	33	590

5.2 User Simulator

Conversational recommender system is a dynamic process of user preference elicitation. Similar to [6, 12, 13, 15, 17], we created a user-simulator to enable the CRS training and testing. This enables us to simulate conversations based on the user-item interactions that we observe in the datasets.

User-item interactions. We use the attributes mentioned in the reviews in the dataset as user-item interaction. For example: in a review $\mathbf{r}_j$ user u describes some attributes $\subseteq \mathcal{F}$ about item i. In each conversational session ("rounds"), an observed user-item pair (u, i) is first selected. We call the item i the target item or the ground truth item for that round.

Limitations of previous simulators. Previous simulators [12, 12, 13] assume that all of item i's attributes $\mathcal{P}_i$ is the oracle set of attributes preferred by the user in this session. This means that any user u will respond in the same way to any specific item. This setting is unrealistic because in reality, every user may not equally care about all the attributes of an item. Hence, this design eliminates the potential of personalized responses.

We design a user-based simulator that can handle user-specific feedback in each conversation rounds. We can simulate a conversation session for each observed interaction between a user and an item in the following way. For any review $\mathbf{r}_j$ about (u, i), our simulator only accepts (responds "Yes" to) an attribute f_l if and only if it is mentioned in $\mathbf{r}$, i.e., $r_{j,l} = 1$; otherwise it will respond "No". For enumerated setting (Yelp dataset), the system asks questions in the format "Which attribute of this category do you like?". For example: "Which kind of coffee do you like?". In this case, user-simulator responds the attribute that was mentioned in the review. Finally, the simulator accepts a TopK recommendation if the target item i appears in the recommendation; otherwise it rejects the recommendation.

Max turn. We set the maximum turn in a conversational round to 15. User leaves the conversation after the turn limit is reached.

Top-K. We set the K = 10, so that we are limited to recommend only 10 items in a recommenda-

tion turn.

5.3 Baselines

To evaluate the efficacy of FACT-CRS, we compare the proposed method with several state-of-the-art CRS methods.

- **Max Entropy (MaxE)** [45]: In this setting, the CRS probabilistically chooses either an attribute to ask or top ranked items to recommend. The CRS asks the attribute with the maximum entropy within the current state. For the enumerated question setting (Yelp dataset), the entropy of an attribute is calculated as the sum of entropy of its child attributes in the taxonomy.
- **EAR** [12]: This is a three stage approach consisting of estimation, action and reflection stages. Estimation stage builds predictive models to estimate user preference on both items and attributes. Action stage learns a dialogue policy to determine whether to ask attributes or to recommend items, based on Estimation stage and conversation history. Reflection stage updates the recommender model when a user rejects the recommendations made by the Action stage. The CRS updates the conversation and recommendation components using reinforcement learning.
- **FPAN** [16]: This is based on the EAR model. It extends the model by using a user-item-attribute graph to learn the offline representation better. They dynamically revise user embeddings based on users' feedback. Relationship between attribute-level and item-level feedback signals are used to more precisely identify the specific items and attributes that causes the rejection of an item. They design two gating modules to adapt the original user embedding and item-level feedback respectively. The gating modules uses the fine-grained attribute-level feedback to modify the user embedding. They also use coarse-grained item-level feedback which enables them to accurately learn user preference.
- **UNICORN** [13]: This integrates the conversation and recommendation components into a unified RL agent. They develop a dynamic weighted graph based RL method to learn a policy to select the action at each conversation turn, either asking an attribute or recommending items. They propose two heuristics for reducing the candidate action space by pre-selecting attributes and items in each turn to simplify the RL training.

All these methods rely on pre-trained user embeddings to make recommendations or construct states, which are not available in new users. To apply them to new users, we used the average embedding of all training users as the embedding for new users in these baselines. All models are evaluated on "cold start" test users.

5.4 Training

We perform a random split of the users for training, validation and testing maintaining the ratio of 80%, 10% and 10%. We make sure that each set contains disjoint set of users. We performed the training of FACT-CRS on training users, and tuned the hyper-parameters on validation users. In this experiment, we set the depth of the user-item interaction tree in FACT-CRS to 5 and the size of latent dimension to 40. The hyper-parameters used in the final training are listed in Table 5.2.

Table 5.2: Hyper-parameters used for training

Hyperparmeter	Description	Value
λ_{s_0}	Regularization coefficient for s_0 for $\mathcal{L}_R$	10^{-2}
$\lambda_{f_{r0}}$	Regularization coefficient for s_0 for $\mathcal{L}_R$	10^{-2}
λ_{v_0}	Regularization coefficient for s_0 for $\mathcal{L}_R$	10^{-2}
$\lambda_{f_{o0}}$	Regularization coefficient for s_0 for $\mathcal{L}_R$	10^{-2}
λ_{BPR}	Weight of BPR loss in SGD objective	10^{-3}
λ_s	Regularization coefficient for s_0 for $\mathcal{L}_R$	10^{-2}
λ_v	Regularization coefficient for s_0 for $\mathcal{L}_R$	10^{-2}
ζ	Stopping criterion for building tree	10^{-5}
α	Negative feedback penalty	10^{-2}
d	Dimension of latent factor	40
H_{max}	Depth of user-item interaction tree	5

5.5 Performance Metrics

Following [6, 12, 13, 15–17], we use success rate and average turn as evaluation metrics.

Success Rate: We use the success rate at turn T (SR@T) to measure the ratio of successful conversations. In a review where user u interacted with item i, we call i the ground truth or the target item. A round of conversation is successful if the model can identify the ground truth item.

Average Turn: We also report the average turns (AT) needed to end the round of conversation. The number of turns in a failed conversation is the max turn T.

The quality of recommendation is greater for larger SR@T. Whereas the conversation is more efficient and on point for smaller AT.

5.6 Comparison

Following the performance metrics in the existing works [12, 13, 17] all models are evaluated on test users with K=10 for top-K recommendations and T=15 for SR@T. We provide additional results for other values of T and K in Figure 5.1 and Figure 5.2.

5.6.1 Overall Performance

Table 5.3: Comparison of CRS performance (K=10)

		MaxE	EAR	FPAN	UNICORN	FACT-CRS	Improvement over baseline*
LastFM	SR@15	0.274	0.413	0.536*	0.453	**0.683**	27.42%
	AT	13.45	11.67	10.13*	10.61	**8.26**	18.62%
Yelp	SR@15	0.863	0.881	0.927*	0.913	**0.946**	2.05%
	AT	6.68	6.38	**5.84**	5.97	6.18	-

We first evaluate the recommendation quality of FACT-CRS in terms of success rate (SR@15) and average turn AT. A good recommender system should be able to realize items which are more relevant to a user's preference and rank them higher in a result list.

We report the results of our experiments in Table 5.3. FACT-CRS consistently outperforms the others by some margin on both LastFM and Yelp datasets. FPAN performs better than the others among the baselines. Although FPAN uses the same general structure as EAR, FPAN updates user embeddings dynamically with users' positive and negative feedback on attributes and items by two gate modules and this enables dynamic item recommendation. That is why FPAN is able to generalize better on the new users. Although EAR is effective in handling large action space in CRS, its performance is limited by the separation of its conversation and recommendation components. UNICORN performs relatively better in this case as it uses a couple of action selection heuristics.

However, since all the models rely on user level embedding, they cannot perform well on new test users. By exploiting the user-item interaction for recommendation, our model builds a tree to learn the user-item interaction embedding, which captures the current preference of the user. Empirically, this approach is found to be superior. FACT-CRS hierarchically clusters the user-item interactions in the tree. The motivation behind it is that the representations of user-item interactions that share the same attributes should be assigned closer to each other. It thus learns the in-group homogeneity better and consequently learns the latent factors better. We can see from Table 5.3 that FACT-CRS achieves a significant improvement in SR@15 by 6.89% against

the best baseline on LastFM and by 2.05% on Yelp. For Average turn, it also improves 18.46% against the best baseline performance on LastFM.

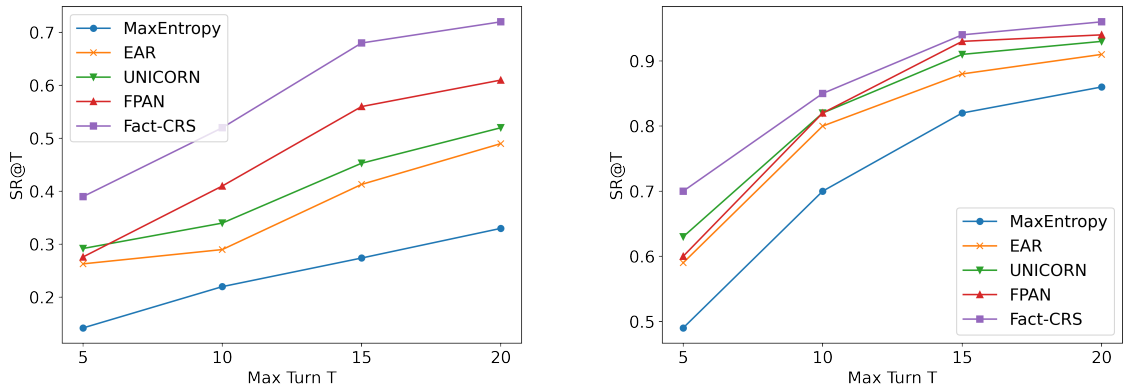

Figure 5.1: Comparison of success rate at different values of Max turn T for (Left) the LastFM dataset and (Right) the Yelp dataset

Figure 5.1 shows the improvement in success rate if we increase the max turn T. For this experiment we set K=10. The performance of all of the methods improve as max turn T increases. In all of those cases, FACT-CRS outperforms the baselines. An interesting thing to note is that even though the depth of the user-item interaction tree is 6, FACT-CRS is still able to recommend at T=5 because every node contains the embedding of the interactions in that node.

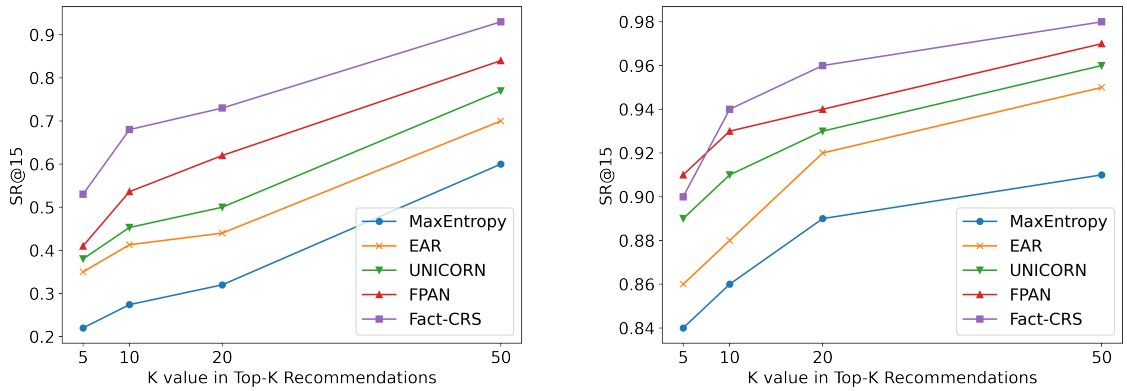

Figure 5.2: Comparison of success rate at different values of K in top-K recommendations for (Left) the LastFM dataset and (Right) the Yelp dataset

Similarly, Figure 5.2 shows the changes in SR@15 by varying K. From our experiments we have seen that the mean rank of the target item on the LastFM dataset is approximately 33. So, all of the methods show improved performance as K increases. FACT-CRS outperforms the baselines for different values of K. We limit the K value at 50 because it is too cumbersome for a real user to go through too many recommendations.

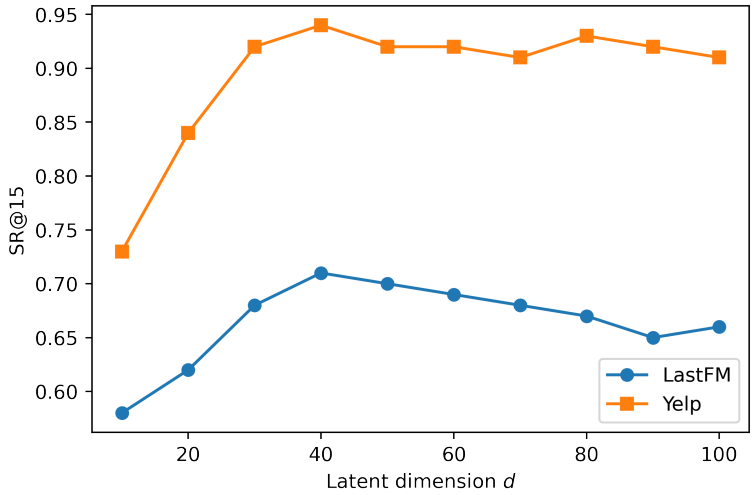

Figure 5.3: SR@15 vs. the latent dimension d

5.6.2 Effect of hyperparameters

Next, we study the effect of two important hyper-parameters in FACT-CRS, which are: the latent dimension and the tree depth.

Latent Dimension d

The dimension of latent factors is an important hyper-parameter for factorization based methods as it determines the model's ability to learn representations. We design this experiment to explore the influence of latent dimension on the performance of FACT-CRS. We vary the dimension of latent factors from 10 to 100 with an increment of 10.

The results are shown in Figure 5.3. It is apparent from the figure that the model couldn't learn the user-item interactions very well when the dimension was small (< 20). And with more latent dimensions (> 50) the performance of FACT-CRS slightly decreases, as higher dimensional latent factor learning needs more training data. This is intuitive as it is more difficult for a model with a lower latent dimension to capture the dynamics between user-interaction and items. However, the models with a higher latent factor dimension usually over-fits without larger training data. We see that on the Yelp dataset increasing the latent factor slightly increases the success rate. For $d = 40$, FACT-CRS has the maximum gain in SR@15. That is why in our experiments we choose latent factor dimension $d = 40$.

Maximum tree depth H_{max}

In the FACT-CRS User-item Interaction Tree, we cluster the interactions (the user-item pair) while building the tree and learn the interaction embeddings. The maximum tree depth has two important roles: 1) It increases the resolution of the user-item interaction embedding 2) It allows the model to ask more questions. This is a trade-off situation where if we ask more questions, we

could potentially get a better user-item interaction embedding but this also means the Average Turn could possibly increase. To find the effect of maximum tree depth, we fix all the other hyper-parameters and vary the maximum depth of user-item interaction tree and look at the changes in the SR@15, as shown in Figure 5.4.

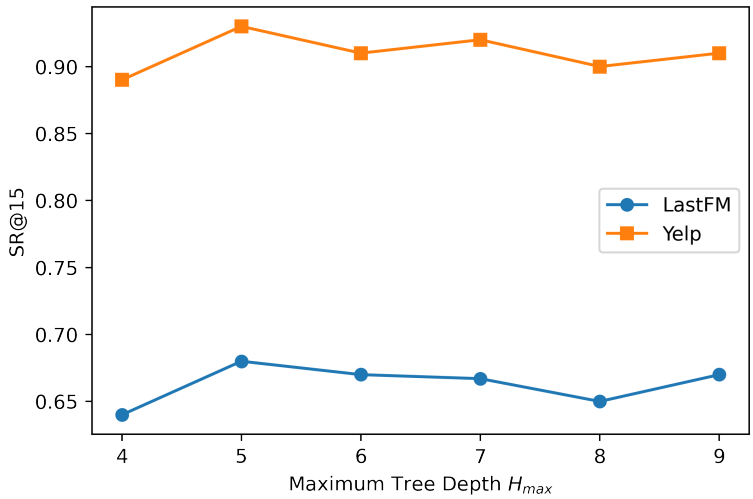

Figure 5.4: SR@15 vs. the maximum depth H_{max} of user-item interaction tree

As we can see in Figure 5.4, FACT-CRS is relatively resilient to the change in depth in the user-item interaction tree. When we initially increase the maximum depth of the tree, the resolution of the user-item interaction embeddings gets better and so we get better performance. However, if we keep increasing the depth (at 6) the performance decreases because the model gets fewer chances to correct itself whenever the user rejects a recommendation.

5.6.3 Negative Feedback: Recommendation Probability and Success Afterwards

We are also interested in how FACT-CRS compares to other baselines when it encounters a rejection. To illustrate this, we compare FACT-CRS's recommendation probability and success rate to FPAN's on LastFM dataset. As shown in Figure 5.5, FPAN can start recommending very early on, but when the user rejects a recommendation, FPAN suffers greatly. For example: after turn 6, the recommendation probability of FPAN is very close to 1. So, FPAN does not have a good approach in identifying where it went wrong. Instead, after a rejection, it tries to recover by just recommending more items until the user quits.

In comparison, FACT-CRS handles the negative feedback in a much better way. It asks further clarifying questions before making another recommendation. Looking at the recommendation success rate, we see that this approach is much superior. After turn 6, FPAN's success rate keeps

getting smaller. At this point, FPAN does not try to improve by asking questions and consequently is not able to identify the reason behind rejection. Whereas, FACT-CRS first tries to identify the negative items and then uses Equation 4.4 to corrects the predicted user-item interaction embedding, and only then moves on to the next tree. This allows FACT-CRS to ask better questions, and make better predictions. As we can see in Figure 5.5, FPAN's success rate after turn 10 is around 0.2 whereas FACT-CRS's success rate is around 0.4. This demonstrates that asking further questions after a rejection and making the correction in the predicted embedding are two important strategies.

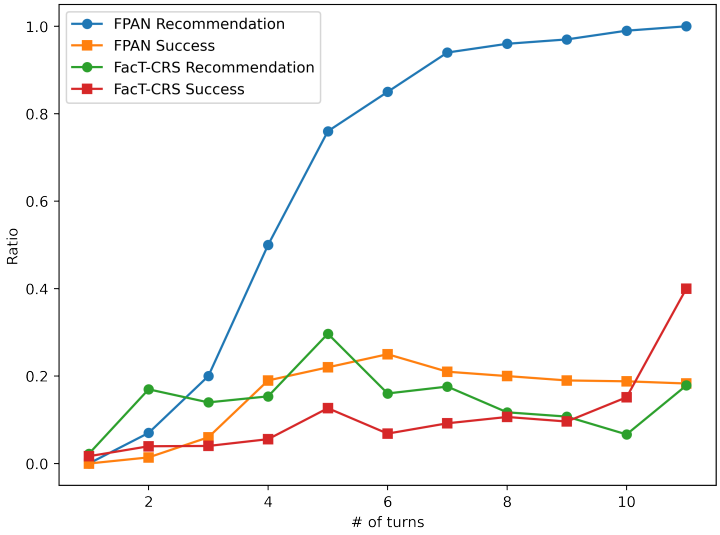

Figure 5.5: Probability of recommendation and recommendation success rate at each turn on LastFM dataset (FACT-CRS vs FPAN)

5.7 Ablation Study

In this section, we are interseted in identifying how much each component in FACT-CRS contributes. We perform ablation study to investigate the effect of each individual component by removing that component and evaluating the remaining model.

5.7.1 Impact of Random Forest

We use random forest to ask any arbitrary number of questions. Also, this is the core concept of how we can extend tree based methods to multi-turn conversations. In this strategy, we sub-sample from the set of all the available features to build individual tree. Without the random forest, the number of questions we can ask at most is the maximum depth of the tree H_{max}.

Table 5.4: Ablation study

	LastFM		Yelp	
	SR@15	AT	SR@15	AT
FACT-CRS	0.683	8.263	0.946	6.18
– Early Recommendation	0.641	9.24	0.921	6.26
– Negative feedback	0.674	8.36	0.940	6.19
– Random Forest	0.283	12.38	0.815	5.24

We can understand how important this component is by removing random forest and building just one tree with all the available features. From Table 5.4, we can see that the random forest based approach is responsible for 50.4% improvement in SR@15 for the LastFM dataset and 16.07% improvement in SR@15 for the Yelp dataset. So this is essentially the main component of FACT-CRS.

5.7.2 Impact of Early Recommendation when Node Contains $\leq$ K Items

In this strategy, at each node in the user item interaction tree, we look at the set of items $\mathcal{I}'_a$. If $|\mathcal{I}'_a| \leq K$, we make an early recommendation. On the cross validation, we have found that even with very noisy user response (i.e., number of mentioned attributes is low and there are many false positive attributes in the interaction), $\mathcal{I}'_a$ is very successful in containing the target item. So, when the number of items, $|\mathcal{I}'_a| \leq K$ we can make a good recommendation. As we can see from Table 5.4, this has significant impact on both the success rate and the average turn. If we remove this strategy from FACT-CRS, SR@15 goes down by 6.15% in the LastFM dataset and by 2.64% in the Yelp dataset. Also, the average turn goes up by almost 1 turn in the LastFM dataset. So, this early recommendation strategy is an important component of FACT-CRS.

5.7.3 Impact of handling negative feedback

As explained in Section 5.8, we have found that in successful recommendations, the target item mostly appears on the leaf nodes. That is why, if recommendation is rejected, we want to correct the predicted user-item interaction embedding to penalize the items not in the leaf node so that those items don't appear on top in the next recommendations. FACT-CRS first tries to identify the items responsible for the rejected recommendations, and corrects the predicted user-item interaction embedding. It then uses the CTF strategy using the corrected user-item embedding. This allows FACT-CRS ask better questions, and make better predictions This is evident from Figure 5.5, as FPAN's success rate after turn 10 is around 0.2 whereas FACT-CRS's success rate

is around 0.4. This strategy contributes to better recommendation and fewer average turn as can be seen in Table 5.4. The improvement in SR@15 is 1.48% on LastFM and 0.64% on the Yelp dataset.

5.8 Case Study

We performed studies to analyze the performance of our model and to identify where we can further improve.

5.8.1 Failed Conversations

We take a special look on the failed conversation to get a better understanding of why a conversation fails. On both LastFM and Yelp dataset, we report the average number of mentioned attributes in the failed interaction and compare it to successful interactions. Table 5.5 summarizes the mean and standard deviation of this results.

Table 5.5: Number of attributes

	LastFM		Yelp	
	Mean	Std dev.	Mean	Std. dev
Successful	4.03	2.46	4.12	2.52
Failed	2.53	1.79	2.14	1.04
All	3.99	2.48	4.06	1.56

As reported in Table 5.5, on both LastFM and Yelp datasets, the average number of mentioned attributes in the reviews of failed conversation is much smaller than the average number of mentioned attributes in all of dataset, and also smaller than the number of mentioned attributes in successful conversations.

On both LastFM and Yelp datasets, the average number of mentioned attributes in the dataset is approximately 4. That is why, we next look at how many of the conversations with at least 4 mentioned attributes are failing. Our experiments show that those cases are relatively rare. In the LastFM dataset, out of all the test reviews which had at least 4 mentioned attributes, only 2.065% has failed. And on Yelp dataset, only 2.01% has failed. So, in conversations where the user can successfully identify at least 4 attributes, the success rate is over 97% on both LastFM and Yelp datasets. This gives us an idea of why the conversations are failing. Since the reviews in the failed conversations are very short, our model does not have enough information to infer which particular item that review is referring to.

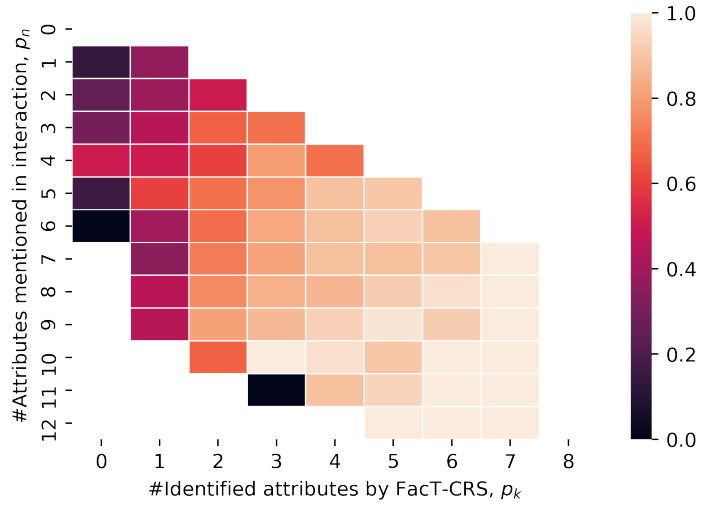

Figure 5.6: SR@15 of the number of attributes identified (correctly asked) by FACT-CRS for different review lengths on LastFM dataset

5.8.2 Identified Attributes

Figure 5.6 shows the success rate of conversations with different review length p_n and number of attributes identified by FACT-CRS p_k. Note that $p_k > p_n$ is not possible, i.e., FACT-CRS cannot identify more attributes than what were mentioned. Also, $p_k < T$, since any CRS agent can at most ask $T - 1$ questions. When $p_k \leq p_n$, the white cells in Figure 5.6 refers to the event not occurring. For example: when $p_n = 9$, FACT-CRS always identified $p_k \geq 1$ attributes. Similarly, when review length was 12, FACT-CRS at least asked 5 attributes correctly and was successful in all of those cases. When more attributes are identified (left to right in Figure 5.6) it is more likely that the conversation will be successful. Similarly, when user mentions more attributes in a review (top to bottom), FACT-CRS is likely to identify more attributes and subsequently the conversations are more likely to be successful.

5.8.3 Impact of Recommendation using User-Item Pairs in Interaction Tree Node

Every node in the user-item interaction tree contains a subset of user-item pairs as member. If we are ready to recommend at a node in the user-item interaction tree (usually at the leaf node), we check which items are in the subset of the user-item pair. Then we rank those items based on the predicted user-item interaction embedding $\mathbf{s}_{pred}$ and the item embeddings $\mathbf{V}$. This enables us to significantly narrow down the candidate set of items. Table 5.6 compares our models performance with this strategy to just using Equation 4.1 to score the items.

This candidate item set is a vital part of FACT-CRS in two ways:

Table 5.6: Effect of user-item pairs in User-item Interaction Tree

	LastFM		Yelp	
	SR@15	AT	SR@15	AT
FACT-CRS	0.683	8.26	0.946	6.18
FACT-CRS (only ranking)	0.524	10.37	0.864	6.58

- As Table 5.6 shows, this is a key concept that makes our recommendation much better by narrowing down the candidate set of items. This strategy alone boosts the performance (SR@15) of our model by about 40% in LastFM and by 9.49% on Yelp dataset.
- The strategies we have discussed in the ablation study, namely early recommendation and negative feedback, rely on the items in the leaf node of user-item interaction tree. So, without this approach, those components would not exist.

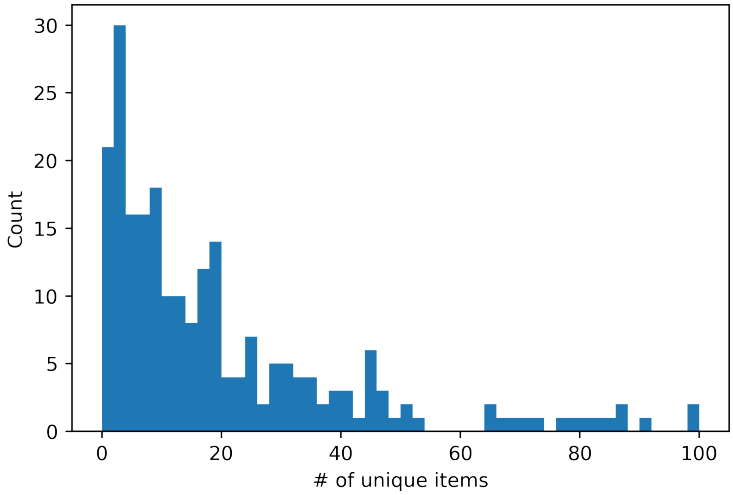

Figure 5.7: Histogram of number of items in leaf nodes in the user-item interaction tree (Last FM)

To understand why using the items in the leaf nodes of the user-item interaction tree is so important, we plot the histogram of the number of unique items in the leaf node on the LastFM dataset. As can be seen from Figure 5.7, more than 90% leaf nodes have very few (< 20) items. So, the leaf nodes work well to cluster and narrow down the correct candidate list of items. This also verifies our initial hypothesis: using the shared attributes to cluster the user-item interaction and subsequently learning the embeddings enhances the quality of the embeddings.

We do another experiment to check how scattered each item is among the user-item interaction tree. We record how many different leaf nodes an item can be found on. If we sort those numbers

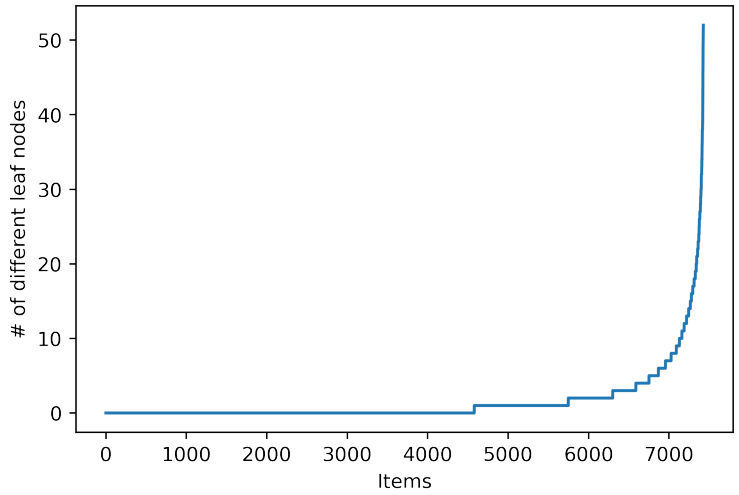

Figure 5.8: Number of different leaf nodes each item appears in the user-item interaction tree (Last FM; in sorted order)

by the count, we get Figure 5.8. This figure demonstrate that most items are not very scattered. In fact, over 85% of the items appear in at most 5 leaf nodes. By combining Figure 5.8 and 5.7, we conclude that using the shared attributes to group the items according to user-item interactions, FACT-CRS can effectively find a good subset of candidate items containing a small number of items.

5.9 Discussion

Our interaction tree based methodology provides both good predictive accuracy as well as minimizing the inputs required from the user. We used "cold start" setting in our experiments to test how FACT-CRS performs on new users. The experiments also show that our method is adaptive to both binary interaction (yes/no answer to the question "do you like this feature?") and enumerated interaction (selecting an answer to the question "Which value of this feature do you like?"). Our model outperforms the existing baselines on both types of datasets, which shows that our model is adaptive to both binary and multi-valued attributes. LastFM dataset contains over 7k items whereas Yelp dataset contains over 70k items. FACT-CRS is effectively able to ask and recommend using both datasets, which demonstrates the scalability of our model. Finally, our model is built using simple decision tree based method and it is able to successfully ask questions to elicit user's preferences, make decisions to recommend and adapt when user rejects recommendation—which are the three challenges in multi-turn CRS. So, we have been able to empirically verify that indeed supervised learning is sufficient for multi-turn CRS.

Chapter 6

Conclusion

In this thesis, we proposed an efficient tree-based algorithm that is able to handle the challenges in conversational recommendations. In Chapter 3 we showed how we can use a decision tree based method, namely FacT to ask questions and recommend. In Chapter 4 we explained our research questions in the FACT-CRS setting and outlined how we can solve those questions.

In this chapter, we present the conclusion of our work. In Section 6.1, we discuss broader implications of our work. Section 6.2 discusses the limitations of our work. The future directions are outlined in 6.3. Section 6.4 presents the summary and final remarks.

6.1 Broader Implications

Social implications. In our daily life we often rely on intelligent assistants such as e.g. Cortana (Microsoft), Siri (Apple), Alexa (Amazon), Google Assistant etc. These assistants are useful when we ask specific questions and expect answer. However, these assistant are limited in the sense that they cannot hold a multi-turn conversation, neither can they ask questions to better understand what the user is looking for. FACT-CRS can be applied in the real world scenarios such as in intelligent assistants/agents. Our model is especially useful where we frequently encounter new users and have to find relevent items just by asking a few questions (e.g., e-commerce/tourism platforms etc.). Our trained model can also be deployed in the devices with limited processing capabilities because it is very generalized, simple and lightweight.

Implications for future research. In addition to empirically demonstrating that rule-based supervised learning is sufficient for multi-turn conversational recommender system, this study also focuses on not relying on pre-trained user embeddings, since multi-turn CRS should be able to create a users profile from online interaction. We give most importance to the user's current need. For example: a user who is looking for a restaurant in a busy area may consider the availability of parking space. On the other hand, a user who is in a hurry may care about

the how much the wait time is. It might be frustrating to users if the CRS do not quickly figure out what the user currently needs, rather relies on user's historical data. FACT-CRS overcomes this problem by taking into account the user-item interaction and learning the latent factor of the interactions. An important objective of our research is providing a new perspective of how we look at the multi-turn CRS problem: we start with simple models, identify the current challenges and check if the model can be modified to meet those challenges.

6.2 Limitations

A limitation of FACT-CRS is that the attributes with high frequency contribute more to the user-item interaction tree. So, when the number of features is very few, FACT-CRS may not be able to create tree branches correctly. A larger set of attributes is helpful for FACT-CRS to learn the interaction between the users and the items. Also, similar to the baselines, the case study on failed conversation showed that FACT-CRS suffers when the number of mentioned attributes in user-item interaction is very few.

There is another component in our design that may apparently seem like a limitation: the number of questions we can ask using a user-item interaction tree is limited to the depth of the tree. However, upon careful observation, we can see that FACT-CRS remembers the previously asked questions and does not repeat them, and that the interaction embedding is corrected when we move on to the next tree. That is how FACT-CRS preserves the context of the conversation.

6.3 Future Directions

Our work is a first in the direction of using supervised learning for multi-turn CRS and using cold start setup. The future direction of this study includes:

Latent Factor Learning Models. In FACT-CRS, we used matrix factorization with Bayesian Pairwise Ranking as a latent factor model. However, there are other latent factor models, such as-factorization machines [36], non-negative matrix factorization [46], incremental singular value decomposition [47] etc. We can also combine various different latent factors to learn a hybrid representation.

Attribute comparison. In our work we use the attribute value to build the tree. For example: for attribute f_l we ask if $f_l > t_l$ where t_l is some threshold, or "do you like attribute f_l?" Alternatively, one can ask the relative ordering of attributes, or comparison of two or more attributes. Is $f_a > f_b$? or "do you like attribute a more than attribute b?" (for example: "do you like coffee more than tea?"). This would allow us to rank the attributes even when we don't know the values of the attributes.

Multi-predicate. We could group together two or more attributes and ask them in the same question. We could use the predicates 'any' (logical or) or 'all' (logical and). For example: is $any(f_a, f_b, f_c)$? or do you like any of attributes a, b, c? Similarly, we could use $all(f_a, f_b, f_c)$? to represent "do you like all of attributes a, b, c?". This would allow us to ask more complex questions and therefore get more information in a single turn.

6.4 Concluding remarks

The current approaches in conversational recommender system rely on reinforcement learning based policy learning. We challenged the necessity of reinforcement learning approach because the multi-turn CRS appears to be addressable just using supervised machine learning. To demonstrate that, we proposed a novel decision tree based approach to CRS. We proposed learning the latent factor of user-item interaction instead of just user or just item latent factors. The intuition behind this approach is, in CRS, the system needs to elicit the current user preference by asking questions and recommending. So, if we can learn the latent factors of the interaction and ask questions on the basis of that, we can effectively obtain user's preferences dynamically. We used FacT [18] to learn the interaction embedding and extended it to meet the 3 main challenges in CRS: 1) what question to ask 2) deciding when to recommend 3) handling negative feedback. We used "cold start" setting in our experiments to demonstrate our model's performance on new users. We extensively experimented on two benchmark CRS datasets, namely LastFM and Yelp, and compared FACT-CRS's performance with existing state of the art models. The experimental results show that FACT-CRS is effectively able to ask and recommend using both datasets. Our interaction tree based method is more successful in recommending the correct target item by 27.42% on LastFM dataset and by 2.05% on Yelp dataset. FACT-CRS is also more effective in inferring quickly using fewer number of question-answer turns. Therefore, we verified that 1) user-item interaction is a powerful method to obtain the user's current preference and 2) the supervised learning is indeed able to meet the challenges in CRS. We hope our work will inspire and help our fellow researchers with the required quantitative information and guideline.

References

[1] D. Booth, "Marketing analytics in the age of machine learning," *Applied Marketing Analytics*, vol. 4, no. 3, pp. 214–221, 2019.

[2] X. He and T.-S. Chua, "Neural factorization machines for sparse predictive analytics," in *Proceedings of the 40th International ACM SIGIR conference on Research and Development in Information Retrieval*, pp. 355–364, 2017.

[3] X. Wang, X. He, M. Wang, F. Feng, and T.-S. Chua, "Neural graph collaborative filtering," in *Proceedings of the 42nd international ACM SIGIR conference on Research and development in Information Retrieval*, pp. 165–174, 2019.

[4] X. Wang, D. Wang, C. Xu, X. He, Y. Cao, and T.-S. Chua, "Explainable reasoning over knowledge graphs for recommendation," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, pp. 5329–5336, 2019.

[5] G. Salton, *The SMART retrieval system—experiments in automatic document processing*. Prentice-Hall, Inc., 1971.

[6] Y. Zhang, X. Chen, Q. Ai, L. Yang, and W. B. Croft, "Towards conversational search and recommendation: System ask, user respond," in *Proceedings of the 27th acm international conference on information and knowledge management*, pp. 177–186, 2018.

[7] F. Radlinski and N. Craswell, "A theoretical framework for conversational search," in *Proceedings of the 2017 conference on conference human information interaction and retrieval*, pp. 117–126, 2017.

[8] J. Chu-Carroll and M. K. Brown, "An evidential model for tracking initiative in collaborative dialogue interactions," in *Computational Models of Mixed-Initiative Interaction*, pp. 49–87, Springer, 1998.

[9] S. Young, M. Gašić, B. Thomson, and J. D. Williams, "Pomdp-based statistical spoken dialog systems: A review," *Proceedings of the IEEE*, vol. 101, no. 5, pp. 1160–1179, 2013.

[10] K. McCarthy, Y. Salem, and B. Smyth, "Experience-based critiquing: Reusing critiquing experiences to improve conversational recommendation," in *International Conference on Case-Based Reasoning*, pp. 480–494, Springer, 2010.

[11] K. Christakopoulou, F. Radlinski, and K. Hofmann, "Towards conversational recommender systems," in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 815–824, 2016.

[12] W. Lei, X. He, Y. Miao, Q. Wu, R. Hong, M.-Y. Kan, and T.-S. Chua, "Estimation-action-reflection: Towards deep interaction between conversational and recommender systems," in *Proceedings of the 13th International Conference on Web Search and Data Mining*, pp. 304–312, 2020.

[13] Y. Deng, Y. Li, F. Sun, B. Ding, and W. Lam, "Unified conversational recommendation policy learning via graph-based reinforcement learning," *arXiv preprint arXiv:2105.09710*, 2021.

[14] D. Kang, A. Balakrishnan, P. Shah, P. Crook, Y.-L. Boureau, and J. Weston, "Recommendation as a communication game: Self-supervised bot-play for goal-oriented dialogue," *arXiv preprint arXiv:1909.03922*, 2019.

[15] Y. Sun and Y. Zhang, "Conversational recommender system," in *The 41st international acm sigir conference on research & development in information retrieval*, pp. 235–244, 2018.

[16] K. Xu, J. Yang, J. Xu, S. Gao, J. Guo, and J.-R. Wen, "Adapting user preference to online feedback in multi-round conversational recommendation," in *Proceedings of the 14th ACM international conference on web search and data mining*, pp. 364–372, 2021.

[17] W. Lei, G. Zhang, X. He, Y. Miao, X. Wang, L. Chen, and T.-S. Chua, "Interactive path reasoning on graph for conversational recommendation," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 2073–2083, 2020.

[18] Y. Tao, Y. Jia, N. Wang, and H. Wang, "The FacT: Taming latent factor models for explainability with factorization trees," in *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 295–304, 2019.

[19] T.-W. Chu and Y.-L. Tsai, "A hybrid recommendation system considering visual information for predicting favorite restaurants," *World Wide Web*, vol. 20, 11 2017.

[20] F. Zhang, N. J. Yuan, D. Lian, X. Xie, and W.-Y. Ma, "Collaborative knowledge base embedding for recommender systems," in *Proceedings of the 22nd ACM SIGKDD Interna-*

tional Conference on Knowledge Discovery and Data Mining, KDD '16, (New York, NY, USA), p. 353–362, Association for Computing Machinery, 2016.

[21] C. He, D. Parra, and K. Verbert, "Interactive recommender systems: A survey of the state of the art and future research challenges and opportunities," *Expert Systems with Applications*, vol. 56, pp. 9–27, 2016.

[22] H. Chen, X. Dai, H. Cai, W. Zhang, X. Wang, R. Tang, Y. Zhang, and Y. Yu, "Large-scale interactive recommendation with tree-structured policy gradient," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, pp. 3312–3320, Jul. 2019.

[23] W. Wu, Z. Guo, X. Zhou, H. Wu, X. Zhang, R. Lian, and H. Wang, "Proactive human-machine conversation with explicit conversation goal," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, (Florence, Italy), pp. 3794–3804, Association for Computational Linguistics, July 2019.

[24] A. Tversky and I. Simonson, "Context-dependent preferences," *Management science*, vol. 39, no. 10, pp. 1179–1189, 1993.

[25] F. N. Tou, M. D. Williams, R. Fikes, D. A. Henderson Jr, and T. W. Malone, "Rabbit: An intelligent database assistant.," in *AAAI*, pp. 314–318, 1982.

[26] B. Smyth and L. McGinty, "An analysis of feedback strategies in conversational recommenders," in *the Fourteenth Irish Artificial Intelligence and Cognitive Science Conference (AICS 2003)*, Citeseer, 2003.

[27] J. Zou, Y. Chen, and E. Kanoulas, "Towards question-based recommender systems," in *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 881–890, 2020.

[28] K. Christakopoulou, A. Beutel, R. Li, S. Jain, and E. H. Chi, "Q&r: A two-stage approach toward interactive recommendation," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 139–148, 2018.

[29] X. Zhang, H. Xie, H. Li, and J. CS Lui, "Conversational contextual bandit: Algorithm and application," in *Proceedings of The Web Conference 2020*, pp. 662–672, 2020.

[30] S. Li, W. Lei, Q. Wu, X. He, P. Jiang, and T.-S. Chua, "Seamlessly unifying attributes and items: Conversational recommendation for cold-start users," *ACM Transactions on Information Systems (TOIS)*, vol. 39, no. 4, pp. 1–29, 2021.

[31] R. Li, S. Ebrahimi Kahou, H. Schulz, V. Michalski, L. Charlin, and C. Pal, "Towards deep conversational recommendations," *Advances in neural information processing systems*, vol. 31, 2018.

[32] X. Chen, S. Li, H. Li, S. Jiang, Y. Qi, and L. Song, "Generative adversarial user model for reinforcement learning based recommendation system," in *International Conference on Machine Learning*, pp. 1052–1061, PMLR, 2019.

[33] Z. Liu, H. Wang, Z.-Y. Niu, H. Wu, W. Che, and T. Liu, "Towards conversational recommendation over multi-type dialogs," *arXiv preprint arXiv:2005.03954*, 2020.

[34] K. Zhou, Y. Zhou, W. X. Zhao, X. Wang, and J.-R. Wen, "Towards topic-guided conversational recommender system," *arXiv preprint arXiv:2010.04125*, 2020.

[35] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, no. 8, pp. 30–37, 2009.

[36] S. Rendle, "Factorization machines," in *2010 IEEE International conference on data mining*, pp. 995–1000, IEEE, 2010.

[37] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme, "Bpr: Bayesian personalized ranking from implicit feedback," *arXiv preprint arXiv:1205.2618*, 2012.

[38] S.-H. Yang, B. Long, A. J. Smola, H. Zha, and Z. Zheng, "Collaborative competitive filtering: learning recommender using context of user choice," in *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, pp. 295–304, 2011.

[39] R. Neches, W. R. Swartout, and J. D. Moore, "Enhanced maintenance and explanation of expert systems through explicit models of their development," *IEEE Transactions on Software Engineering*, no. 11, pp. 1337–1351, 1985.

[40] J. Dougherty, R. Kohavi, and M. Sahami, "Supervised and unsupervised discretization of continuous features," in *Machine learning proceedings 1995*, pp. 194–202, Elsevier, 1995.

[41] S. C. Johnson, "Hierarchical clustering schemes," *Psychometrika*, vol. 32, no. 3, pp. 241–254, 1967.

[42] C. Gini, "Measurement of inequality of incomes," *The economic journal*, vol. 31, no. 121, pp. 124–126, 1921.

[43] T. Bertin-Mahieux, D. P. Ellis, B. Whitman, and P. Lamere, "The million song dataset," 2011.

[44] "Yelp dataset challenge," 2015.

[45] J. Wu, M. Li, and C.-H. Lee, "A probabilistic framework for representing dialog systems and entropy-based dialog management through dynamic stochastic state evolution," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 23, no. 11, pp. 2026–2035, 2015.

[46] P. Paatero and U. Tapper, "Positive matrix factorization: A non-negative factor model with optimal utilization of error estimates of data values," *Environmetrics*, vol. 5, no. 2, pp. 111–126, 1994.

[47] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Incremental singular value decomposition algorithms for highly scalable recommender systems," in *Fifth international conference on computer and information science*, vol. 1, pp. 27–8, 2002.