Learning-to-Suggest: Product Recommendation via Several Questions

Xianchao Wu
A.I.&Research, Microsoft Development Co., Ltd.
xiancwu@microsoft.com

1 Introduction

In this paper, we propose a *learning-to-suggest* (LTS) framework for product recommendation through chatbots via asking the user several questions. We focus on two scenarios: general gift recommendation and specific keyword-based recommendation. For general gift recommendation, the user is supposed to know for whom to buy (such as family members) but do not know what to buy. Then, the chatbot's task is to construct a gift-receiver profile through asking several questions which are related to gender and hobbies. This is more alike "mind-reading" based recommendation. For specific keyword-based recommendation, the use knows what to buy (such as shoes) or what to book (such as hotel rooms) but need detailed selection from candidate categories, materials and attributes of the target candidate products related to those keywords. Then, the chatbot's task is to construct a product profile beforehand and ask several questions related to hierarchical category, material or attribute information (illustrated in Figure 1).

Traditional search engine or collaborative filtering based methods are not well suitable for these scenarios. The first is that search engines will always need keywords for keyword-product matching. The second is that collaborative filtering will make a recommendation based on other/same people's buying history. However, we prefer a user-friendly way of first talking to the user and then making a suggestion based on asking questions related to mind-reading or product categories/attributes. Note that there are gaps of information richness of between the chatbot and the users. The users are supposed to be unfamiliar with the details of the products and they need a user-friendly way to obtain these information. These motivated our LTS framework for chatbots aiming at user-specific and user-friendly product recommendation. The novel ideas in our framework include: (1) we dynamically detect user's intentions of gift buying or specific keyword based product filtering; (2) we attach selectable options to the questions, so that the role of users is switched from "requirement information supplier" to "option learner and selector" and (3) we propose entropy-based and reinforcement learning based methods for ranking questions based on user's answering history. Our target is to recommend the optimal products while asking as few questions as possible.

U: Help me find a gift
A: Sure, who are you shopping for?
  1.family; 2.friend; 3.coworker
U: 2
A: what do your friend do frequently?
  1.playing outside; 2.reading; 3.TV watching

U: any shoes?
A: Sure, 1.sports; 2.casual; 3....?
U: 1
A: is it for 1.pure running; 2.basketball; 3....?

A: Hi there, how can I help you? 1.hotel, 2.hospital,
  3.restaurant, 4.train, 5.attraction
U: 1
A: Okay, do you have a specific area you want to stay in?
  1.centre, 2.east, 3.west, 4.south, 5.north
U: 1
A: Which price range do you prefer?
  1.economic, 2.business, 3.luxury
U: 1

Figure 1: Three examples under our LTS framework: general gift recommendation (top), keyword-based for shoes (middle) and for hotel booking based on Multi-WOZ dataset (bottom).

2 Learning-to-Suggest Framework

Our LTS framework manages chat flow, dialog state and slot filtering through leveraging the 20-question-game idea (Zhao and Eskenazi, 2016; Chen et al., 2018; Hu et al., 2018; Wu et al.,

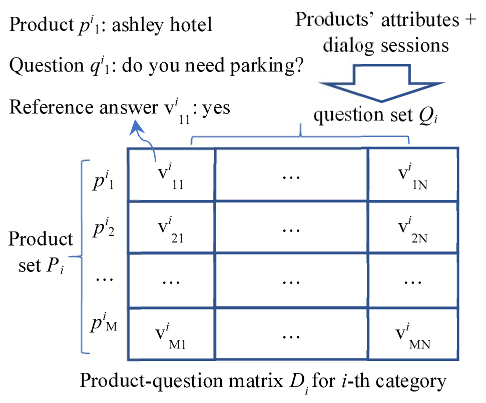
Product p^i_1: ashley hotel
Question q^i_1: do you need parking?
Reference answer v^i_{11}: yes
Products' attributes + dialog sessions
question set Q_i
Product set P_i
p^i_1
p^i_2
...
p^i_M
v^i_{11}
v^i_{1N}
v^i_{21}
v^i_{2N}
v^i_{M1}
v^i_{MN}
Product-question matrix D_i for i-th category

Figure 2: The matrix that links *products* and *questions*.

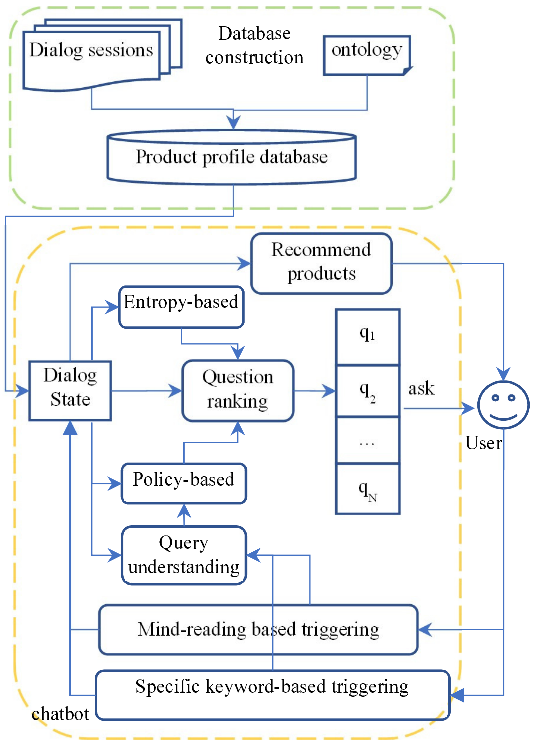

Figure 3: Our LTS framework as a question chatbot.

2018). Figure 2 shows a reference answer matrix D_i which is designed to link the product set P_i and the question set Q_i for each category i (e.g., shoes, hotel). One element in D_i is denoted as v^i_{mn} in which m is the index for a product p_m and n is the index for a question q_n. The question set Q_i comes from the attributes and their values of products.

The LTS framework (Figure 3) includes two modules: (1) constructing the database of P, Q and their relation matrices D and (2) applying the database through a chatbot for query understanding (by slot filtering), question ranking, user's response processing and candidate product recommending.

3 Entropy-based Question Selection

The entropy-based question selection method was described in (Wu et al., 2018; Hu et al., 2018) for famous people guessing. In this paper, we adapt this method to the field of product recommendation. The major idea is to select the next question that can prune as many candidate entries as possible, no matter the user's answer is *yes* or *no*.

Initially, for one category i, we assign a prior popularity weight $w(\cdot)$ to each candidate product p^i_m $(1 \leqslant m \leqslant M)$. The weight can comes from (1) search frequency in search engine, (2) review score in e-commerce websites or even (3) manually constructed dialog sessions as adopted in this paper. Then, we normalize $w(\cdot)$ by:

$$w'(p^i_m) = \frac{w(p^i_m)}{\sum_m w(p^i_m)}$$

For one candidate p^i_m, we compute its contribution $Y^l_{m,n}$ of selecting a question q_n:

$$Y^l_{m,n} = \frac{f^l_{m,n} + \alpha I(v^{i,l}_{m,n} = yes)}{\sum_{l=1}^{L_n} \{f^l_{m,n} + \alpha I(v^{i,l}_{m,n} = yes)\}} \quad (1)$$

Here, $f^l_{m,n}$ stands for the frequency that users selected option l of q_n for a final chosen of p^i_m. $I(\cdot)$ stands for an indicator function that returns 1 when $v^{i,l}_{m,n}$ equals to *yes* and 0 otherwise. We introduce a parameter α here to balance users' selections and the reference answer included in the product-question matrix D_i.

When we range over all candidate p_m, we obtain Y^l_n which stands for the importance of option l in question q_n:

$$Y^l_n = \sum_{m=1}^{M} w'(p^i_m) \times Y^l_{m,n}$$

We set the negative variance of Y^l_n for ranking q_n:

$$w(q_n) = -\sum_{l=1}^{L_n} (Y^l_n - \frac{\sum_{l=1}^{L_n} Y^l_n}{L_n})^2$$

A slightly different way is to use the negative Shannon entropy for the Multivariate Bernoulli distribution of options:

$$M_{m,n} = \sum_{l=1}^{L_n} Y^l_{m,n} log_2 Y^l_{m,n}$$

Then, the weight of a question q_n is defined as:

$$w'(q_n) = \sum_{m=1}^{M} w'(p^i_m) \times M_{m,n} \qquad (2)$$

We skip the category index i for most variables except p^i_m for simplicity.

4 Policy-based Reinforcement Learning

Policy-based reinforcement learning algorithms have been used in (Hu et al., 2018; Chen et al., 2018) for 20-question games such as guessing of famous people with limitations of only allowing game players to answer *yes*, *no*, or *unknown* for the single-attribute questions. We adapt this method to suitable to multiple option attached questions for product recommendation.

Alike the original 20-question game, we formulate the process of question ranking as a finite Markov Decision Process (MDP) expressed by a 5-tuple $\langle S, A, P, R, \gamma \rangle$, where S is the continuous dialog state space, $A = \{q_1, ..., q_n\}$ is the set of all available actions (i.e., questions in our scenario), $P(S_{t+1} = s'|S_t = s, A_t = q)$ is the state transition probability matrix, $R(s, q)$ is the reward function and $\gamma \in [0, 1]$ is a decay factor to discount the long-time returns. In our policy-based reinforcement learning algorithm, at each time-step t, the question chatbot takes an action q_t under the current state s according to the policy function $\pi_\theta(q_t|s)$. After applying q_t to s and receiving user's answer to q_t (i.e., interacting with the environment), the question chatbot receives a reward score r_{t+1} and the dialog state updates from s to s'. This quadruple $\langle s, q_t, r_{t+1}, s' \rangle$ is taken as an episode in the reinforcement learning process.The long-time reward R_t of time-step t is traditionally defined to be $R_t = \sum_{k=0}^{T-t} \gamma^k r_{t+k+1}$.

In our LTS framework, a dialog state s_t keeps track of time t's confidence of candidate products $\{p^i_m\}$ to be sent to the user. Specifically, $s_t \in \mathbb{R}^M$, $\forall s_{t,m} \geq 0$ and $\sum_{m=1}^{M} s_{t,m} = 1$. Here, $s_{t,m}$ denotes the confidence that product p^i_m is user's prefer at time-step t. Initially, s_0 can take the prior distribution of candidate products as we described in the entropy-based question ranking method.

Given the product set $P_i = \{p^i_m\}$ and the question set $Q = \{q_n\}$, we compute the normalized confidence of user's answer over the optional candidates attached in each question q_n. That is, the transition of dialog state is defined as:

$$s_{t+1} = s_t \odot \beta.$$

category	# product	# question (combined)	# avg. *yes*
attraction	79	119 (2)	4.5
hospital	66	88 (0)	2.7
hotel	33	43 (6)	7.2
restaurant	110	146 (2)	4.4
train	2,828	31 (5)	7.0

Table 1: Statistics of products and questions in the five categories of the Multi-WOZ corpus.

Here, $\odot$ is the dot product operator, β depends on the user's answer (selection) x_t to the question q_t (with a index of n_t in the question set $\{q_n\}$) which is selected by the question chatbot at time-step t. We define $\beta = [Y^{\{l\}}_{1,n_t}, ..., Y^{\{l\}}_{M,n_t}]$ when user selected option(s) $\{l\} \subset [1, ..., L_{n_t}]$ for current question q_t and $Y^{\{l\}}_{m,n_t} = \sum_{l \in \{l\}} Y^l_{m,n}$ takes a similar definition in Equation 1. Through this way, the confidence $s_{t,m}$ of a candidate product p^i_m is updated to $s_{t+1,m}$ based on user's answer $\{l\}$ to the selected question q_t at time-step t.

We further accept specific keyword based requirements (refer to the bottom part in Figure 3). The keywords (i.e., slots) included is first detected by existing single sentence oriented slot filtering methods (Mesnil et al., 2013). Then, the questions that are related to the slots are retrieved and their values are assigned. Suppose at time-step t, we receive answers $\mathcal{L}_t = \{\{l\}^1, ..., \{l\}^u\}$ for a list of questions $\mathcal{Q}_t = \{q^1_t, ..., q^u_t\}$. We then need to update from s_t to s_{t+1} by applying all the answers to the questions and respectively updating s_t:

$$s_{t+1} = s_t \odot \beta^1 \odot \beta^2 ... \odot \beta^u \qquad (3)$$

That is, the confidence of a candidate product is continuously updated until we range over all the u questions and their answers.

5 Experiments

5.1 Setup and Results for Multi-WOZ

We choose the Multi-WOZ corpus (Budzianowski et al., 2018) to testify our LTS framework. We pick five domains *attraction*, *hospital*, *hotel*, *restaurant*, and *train* since the number of products and their attribute ontology are in a reasonable size.

We use $\mathscr{R}$ to denote the LTS model in which the reward net is updated jointly with the policy model. In contrast, we use $\mathscr{R}_0$ to denote the non-joint model which only returns rewards alike the entropy based method.

	entropy	$\mathscr{R}_0$	$\mathscr{R}$	$\mathscr{R}$+ dialog sessions
attraction	44.7	52.7	69.9	-
hospital	70.5	82.7	93.1	-
hotel	72.0	75.4	88.9	91.9
restaurant	43.3	37.5	61.1	63.9
train	59.2	64.2	73.4	-

Table 2: Comparison of succeed rates (%) evaluated by self-play (at epochs = 50,000), across categories and method configurations.

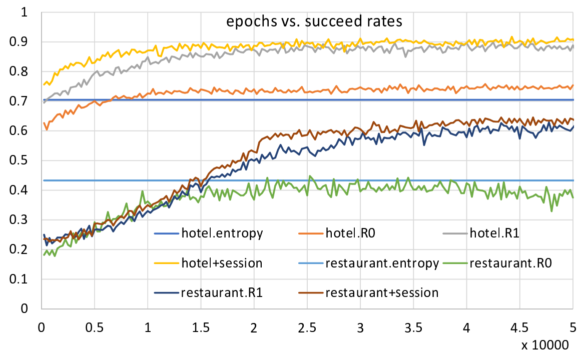

Figure 4: LTS's policy-learning curve of accuracy conditioned on epochs for hotel and restaurant domains. R0=$\mathscr{R}_0$, R1=$\mathscr{R}$.

Figure 4 shows the learning curve of accuracy conditioned on epochs (maximum 50,000) for hotel and restaurant domains under three configurations. The entropy-based method is drawn as well for a reference. For the restaurant domain, the entropy based method outperforms the reinforcement methods before 15,000 epochs, after that, the reinforcement learning method significantly improved with more learning epochs. For the hotel domain, the joint method $\mathscr{R}$ achieves comparable accuracy even at the beginning guided by the product-question matrix. In these two domains, we find that $\mathscr{R}$ is robust no matter there are dialog sessions for pre-training or not.

5.2 General Gift and Keyword-based Recommendation

Besides the service booking recommendation, we further launched two product recommendation systems, general gift and keyword-based recommendations. We collected 500 products under 50 categories as gifts and collected 50 combined questions related to gender and hobby information. We manually link the categories of the products to the questions and each question has averagely around 10 categories assigned to be related. For example, *running shoes* for people who prefer to go outside exercises. Our self-playing results yields an accuracy of 24.5% with averagely 8 questions.

For the keyword-based recommendation, we collected 1,000 products with 20 categories and 2,000 keywords, our self-playing results reflects an accuracy of 85% for the top-3 recommendations with averagely 5 questions.

6 Conclusion

We have described an end-to-end LTS framework for *question chatbot* constructing aiming at product recommendation. One "end" is product-attribute tables and the other "end" is multi-domain question chatbots. Our framework is suitable for situations where only product-attribute tables are available and there are no or quite few real-world dialog sessions belong to that target domain. By constructing a question chatbot, we switch the terminal users of from requirement suppliers to simple decision makers for both time saving and usage-threshold lowering.

References

Paweł Budzianowski, Tsung-Hsien Wen, Bo-Hsiang Tseng, Iñigo Casanueva, Stefan Ultes, Osman Ramadan, and Milica Gasic. 2018. Multiwoz - a large-scale multi-domain wizard-of-oz dataset for task-oriented dialogue modelling. In *Proceedings of EMNLP*, pages 5016–5026.

Yihong Chen, Bei Chen, Xuguang Duan, Jian-Guang Lou, Yue Wang, Wenwu Zhu, and Yong Cao. 2018. Learning-to-ask: Knowledge acquisition via 20 questions. In *Proceedings of ACM SIGKDD*, pages 1216–1225. ACM.

Huang Hu, Xianchao Wu, Bingfeng Luo, Chongyang Tao, Can Xu, wei wu, and Zhan Chen. 2018. Playing 20 question game with policy-based reinforcement learning. In *Proceedings of EMNLP*, pages 3233–3242.

Grgoire Mesnil, Xiaodong He, Li Deng, and Yoshua Bengio. 2013. Investigation of recurrent-neural-network architectures and learning methods for spoken language understanding. In *Interspeech 2013*.

Xianchao Wu, Huang Hu, Momo Klyen, Kyohei Tomita, and Zhan Chen. 2018. Q20: Rinna riddles your mind by asking 20 questions. In *Proceedings of (Japan) NLP*, pages 1312–1315, Okayama.

Tiancheng Zhao and Maxine Eskenazi. 2016. Towards end-to-end learning for dialog state tracking and management using deep reinforcement learning. In *Proceedings of SIGDD*, pages 1–10, Los Angeles.