Soup ADVISER

Kuan-Yu Lin **Tanishq Quraishi**
Institut für Maschinelle Sprachverarbeitung
University of Stuttgart
{st180665,st180925}@stud.uni-stuttgart.de

Abstract

This paper contains the documentation to build the Soup ADVISER. The domain choice is recipes. It is a simple and effective dialog system that helps users find time and ingredient-sensitive recipes to make quick and easy soups. The ADVISER 2.0 toolkit(Li et al., 2020), developed at the University of Stuttgart, has been implemented. The divergence in data structure between the Soup ADVISER and ADVISER 2.0 toolkit causes particular challenges in the implementation that are discussed. For future work, Soup ADVISER can include more recipes beyond soups simply due to the nature of the domain. Users can have a broader range, thus proving the scalability of the task at hand. The code of this project has been released on this repository.

1 Introduction

Dialog systems, also called chatbots, are developed and widely used in various fields in recent years(Adamopoulou and Moussiades, 2020). They aim to solve problems in a way that imitates human conversation. Interacting with dialog systems is no different from having people-oriented solution development. Popular chatbot formats may be found across several industries. Users find dialog systems while booking flights or hotels online, or in the initial phases of customer care and relations. Large portions of customer service have been automated with the inception of dialog systems.

Dialog systems are helpful in many different ways, for instance, in business, information retrieval, e-commerce, etc(Shawar and Atwell, 2007). If dialog systems are built to solve daily problems that most people have, they can be profitable to numerous people, not just limited to certain users. Making food can be a daily problem. The motivation of this project lies in solving a small but everyday problem. People often have insufficient time or ingredients to make food. They look for recipes with the aforementioned constraints in mind. Therefore, we introduce Soup ADVISER which aids users in finding soup recipes.

Soup ADVISER is a task-oriented dialog system, equipped to provide information about the recipe domain. The database includes soup names that occur as entities. The properties of these entities are ingredients, cooking time, and cooking steps. The goal of Soup ADVISER is to provide a soup recipe suitable for the user. On one hand, Soup ADVISER is enabled to gain information that the user provides. On the other hand, it is capable of making requests to the user to retrieve more significant information.

The user can provide the information to Soup ADVISER in several ways. The user can directly ask for a specific soup with its name. For example, 'I want chili soup.' As an alternative, the user can provide the time they have at hand or mention specific ingredients, for instance, 'I have an onion.' This information is then mapped to the database to output valid recipe options.

Once Soup ADVISER gathers enough information and picks a certain soup recipe, the user can further request for cooking time, the ingredients required, and the cooking steps. All recipes contain three cooking steps and the user can not only ask for a specific step but also ask for the previous step or the next step. In this specific situation, if Soup ADVISER answers with 'the first step' or 'the second step' in the previous dialog turn, then the user can ask for the next step and vice versa.

Soup ADVISER is designed based on ADVISER 2.0(Li et al., 2020) toolkit(ADVISER). There are constraints while implementing the ADVISER with different domains. The data of soup recipes include cooking time, cooking steps, and the number of ingredients. In the following sections, we highlight key issues such as values in numbers, the decision-making process of ADVISER when a user queries out-of-order steps, and the distinct data structure of ingredients. We made appropriate modifications to adapt ADVISER 2.0 toolkit with the recipe domain. Solutions to the challenges we have encountered are briefly described.

Firstly, the original ADVISER only accepts data values as text type. Soup ADVISER needs to understand numbers because the value of time and amount are numerical. Secondly, the ADVISER stores the details, (that is the action choice), of each dialog turn, but not the entire dialog. This may pose an issue when the user asks for the last or next step. Soup ADVISER requires the information of its last output to decide what the correct follow-up system act should be. Lastly, the data structure of ingredients is different from other types of data. We list the information of each ingredient separately, instead of grouping them into one slot, named 'ingredient'. In this case, the way to retrieve the data of ingredients is distinct from others, such as time, or cooking steps.

2 Methods

In this section, the source and the structure of the data is described. We explain how we apply the soup recipe domain to ADVISER.

2.1 Database

Recipes were sourced from Comfortable Food[1] and adapted to meet the requirements for the project. For instance, certain recipes contained several micro steps that were merged into the three step process. Recipes with little to no common ingredients were overlooked in the interest of the size of the project.

Database development process commenced by compiling easy soup recipes with all the information found; including items in the following list:

1. The name of the soup

2. The sum of preparation time and cooking time: It is unlikely that users will look up a recipe based specifically on preparation time or cooking time. They provide the total time. Therefore, we use the sum of preparation time and cooking time presented in the recipe. We retained the split in this information within the database so as to remain true to the nature of the recipe domain.

3. The steps of the cooking process

4. The ingredients: Each ingredient separately contains binary values (true or false) indicating presence or absence in a recipe.

The database was created in SQLite.

2.2 System

Natural Language Understanding (NLU) A rule-based natural language understanding (NLU) is part of the ADVISER toolkit. It maps the user utterance to a machine-readable representation for Soup ADVISER. The representations include 'intent', 'slot', and 'value'. Intent presents the action that the users want to execute, for example, 'greet', 'inform', 'request', etc. The slot contains particular information that the user seeks, for example, 'time', 'garlic', etc. Value obtains the value of the information, for example, '10' in the time slot represents 10 minutes; '1' in the garlic slot represents the presence of garlic in the recipe.

Belief State Tracking (BST) One Belief State Tracking (BST) entity is established in the dictionary data structure. BST tracks user action and the information that the ADVISER retrieves from the data in detail for each dialog turn. It is responsible for maintaining a record of the conversation with the user. The role of the BST is primarily memory. BST is renewed after each dialog turn.

Dialog Policies The main purpose of Dialog Policies is to decide the appropriate system action for ADVISER. Though ADVISER 2.0(Li et al., 2020) toolkit introduces several types of policy, we implement a handcrafted policy to Soup ADVISER. According to the rules in the policy and all the information given by the users, Soup ADVISER can find the most suitable entity. Furthermore, Soup ADVISER can also present further information on the entity.

Natural Language Generation (NLG) The

[1]https://comfortablefood.com/35-easy-soup-recipes-with-few-ingredients/

Natural Language Generation maps the machine representation, that is, the information Soup ADVISER found for the users, to natural language. We implement the template-based NLG from ADVISER 2.0(Li et al., 2020) toolkit. The handcrafted template assures that the Soup ADVISER produces coherent and human-like sentences.

3 Challenges and Solutions

In this section, we specify three key issues that were tackled by modifying the original ADVISER 2.0 build.

3.1 Time

There are several ways to express the time such as 1 hour or 60 minutes. Soup ADVISER accepts both, but the time value is only present in minutes in the database. Besides, value of time is acceptable in any number. But, there are only specific time values based on individual recipes in the data. Soup ADVISER should ideally be able to find the recipe without the exact time value match provided by the user.

The rules to match the value of time to the correct data in our domain are added in the NLU. One rule is added to convert hours to minutes. The purpose of this rule is to allow users to freely convey their time constraints.

In order to find the best match between the time that the user has and the time in the recipe, a complementary rule is utilized to map a user's preference to the list of time values in all the recipes. For example, there are only three recipes with the cooking time 10 minutes, 15 minutes, and 20 minutes in the Soup ADVISER. If the user conveys that they have about 18 minutes on hand, there are two possible recipe options: 10 minutes and 15 minutes. Soup ADVISER opts for the 15 minute recipe, which entails the maximum cooking time of all possible recipes.

3.2 Last and Next Step

The definition of 'last step' and 'next step' is based on the content in the previous dialog turn. Only when Soup ADVISER responds to the cooking steps in the previous dialog turn, is it then meaningful for the user to request 'last step' or 'next step'. However, ADVISER 2.0 toolkit does not store all system responses. Also, 'last step' and 'next step' are not slot names present in the data (for the system to retrieve a value from). Accordingly, in BST, we add a new empty list to record the history of the system response and a rule to replace 'last step' and 'next step' to the step slots we have in the data.

When the user requests for 'last step' or 'next step', the system firstly check the last system response in the history, and then, it decides whether to respond or not. If the system accepts the user request, the newly added rule replaces 'last step' or 'next step' with the corresponding step slots. Hence, the system can properly answer the user's request. For instance, if the user asks for 'next step' and the last system response is the first step of the recipe, then the system will renew the BST with the request as the second step of the recipe. Thus, Soup ADVISER can retrieve appropriate information for the user.

3.3 Ingredients

The information of ingredients in the recipe is documented in the slots, named with each individual ingredient, with a binary value. For example, 'Easy Chili' recipe contains tomatoes but no eggs. The value of slot 'tomato' is '1', and the value of slot 'egg' is '0'. This data structure poses to two issues to the accuracy of system responses.

The first issue is to respond to queries such as 'do I need "name of specific ingredient"'- it is a user request seeking further details about the recipe. If the system simply replies with the value, the sentence is incorrect. In the example of 'Easy Chili', if the user asks 'do I need tomato', the system response is 'You need 1.' The system response to 'do I need eggs' will be 'You need 0.' Not only are the objects in the responses absent, but the sentences are also meaningless. Therefore, the rules in the policy are altered to fit our data for this purpose. The system thus responds with 'You need 1 tomato' for the user request about tomato and states 'you need no egg' as a response to the user request regarding eggs.

The second issue is with regards to the query 'what do I need". In a request such as this, the user wishes to obtain all the ingredients that are included in the recipe. This is a user request for 'ingredient', which is not a slot in the data that has a value for the system to retrieve. Further, each recipe needs different number of ingredients. The system responses vary depending on it. We modified several parts to deal with this issue. In policy, the rule is set to examine all the value of the slots which belong to ingredients. Next, the name of slots, which is also the name of the ingredients, with value in '1' are kept in a list. Finally, a rule is created to match the number of types of ingredients to the suitable system response in NLG. Namely, to respond that a certain recipe needs tomato, tofu, and egg, the system response will be 'You need tomato, tofu, and egg to make this soup.'

4 Limits

A major limitation to Soup ADVISER occurs when the data is expanded. Several hard-coded sections need to be altered, such as the list of time value and the list of name of ingredients. The solutions we did to solve the issues in time and in ingredients includes the handcrafted lists. Their size needs manually expansion along with growth of the size of data.

In addition, the rule of 'last step' and 'next step' mapping only works for the recipe with three cooking steps. If a recipe with more than three cooking steps is included in the data, the rule requires modification to be able to function.

Another limitation present is when Soup ADVISER collects the information from the user and chooses the best match recipe. If the users want to find a recipe that includes the ingredients they have, they usually inform all of them to the system. Soup ADVISER only chooses the recipe if it needs every ingredient that the user informs. Informing more or one less ingredients causes failure in finding a recipe.

5 Future work

In Soup ADVISER, the data of ingredients are introduced with binary values. In the future, the binary value can be altered into integers to indicate the number of ingredients that the recipe needs. For example, if the slot 'onion' for 'tomato soup' has '3' as the value, it indicates that tomato soup requires three onions. The unit can also be added to make the information more clear, such as '300 grams onion'

The usage of numbers in the value makes the data scalable. Another possible expansion is to add an adjustment for the recipe depending on the size of the dish. Originally, the recipe in the data is a one-person dish. The adjustment to the amount of ingredients enables users to request the recipe for two or more people.

This project is viable not just for soups, but for recipes in general. Standard recipe structures are followed on most websites which makes data sourcing convenient. All other entities will contain the same properties. It is noteworthy to mention that recipes generally provide optionality. For instance, swapping one ingredient for another, several open-ended choices for garnishing, and more. This feature can be leveraged in the database creation process.

6 Credits

The authors would like to extend their gratitude to their supervisor Lindsey Vanderlyn for her continuous support, advice and feedback.

References

Eleni Adamopoulou and Lefteris Moussiades. 2020. An overview of chatbot technology. In *IFIP International Conference on Artificial Intelligence Applications and Innovations*, pages 373–383. Springer.

Chia-Yu Li, Daniel Ortega, Dirk Väth, Florian Lux, Lindsey Vanderlyn, Maximilian Schmidt, Michael Neumann, Moritz Völkel, Pavel Denisov, Sabrina Jenne, et al. 2020. Adviser: A toolkit for developing multi-modal, multi-domain and socially-engaged conversational agents. *arXiv preprint arXiv:2005.01777.*

Bayan Abu Shawar and Eric Atwell. 2007. Chatbots: are they really useful? In *Ldv forum*, volume 22, pages 29–49.