The Blue Ridge Parkway Tour Assistant

Zihao Ye
Georgetown University
abeyezihao@gmail.com

Abstract

This paper discusses the system architecture and design approaches of the Blue Ridge Parkway tour assistant, which is a dialogue system built on the MindMeld conversational AI platform and discusses the objective and subjective methods used to evaluate this system.

1 Introduction

The *Blue Ridge Parkway* is a National Parkway and All-American Road in the United States, noted for its scenic beauty. The parkway, which is America's longest linear park, runs for 469 miles (755 km) through 29 Virginia and North Carolina counties, linking Shenandoah National Park to Great Smoky Mountains National Park.[Foundation()] It is a place of various and significant natural resources, and its 496 miles offer many ways to explore rich traditions. You can imagine, as a visitor, it will be difficult to find the information you want on their official website in a short time, if you want to make a travel plan. That is why I want to design a tour assistant, which users can ask any questions about the Parkway directly to, instead of searching on the website page by page. Additionally, it seems that there are no existing dialogue systems focusing on parkways or national parks, therefore this system might become a good template for other developers to design their tour assistant system.

Functionally, this system should be able to answer users' various questions related to attractions, trails, food, hotel, and even some specific questions about gas station, speed limit, etc. An example dialogue is shown in Table 1, and more design details will be discussed further below.

. . . greeting . . .
U: Are there any trails along the parkway?
S: (list all the trails)
U: What is the length of the mountain farm trail?
S: Its length is 0.25 mile.
U: Is it difficult?
S: Its skill level is easy.
U: What is the speed limit on the parkway?
S: Maximum is 45 miles per hour, with some locations (in congested areas such as Mabry Mill) at 25 miles per hour.
U: where can I find the latest information on Parkway detours and closings?
S: The most up to date closing information about the Blue Ridge Parkway can be accessed by viewing the Real-Time Road Closure Map: go.nps.govbrp-map
. . . exit . . .

Table 1: A dialogue example.

2 Related Work

I found a MindMeld-based tour assistant at `https://github.com/HimanshuMittal01/tour-assistant-mindmeld` on GitHub, which unfortunately is too simple to build on. So I just started building the application on a basic template provided by MindMeld.

To better answer the user's questions,

this system needs a database to store the parkway-related information, however, no related dataset was found online. Therefore, I just manually collected the data of FAQ, lodgings, restaurants, trails and overlooks from the official website and some other websites about the parkway, instead of using an extra trail API or an restaurants API. I noticed that the restaurants listed on the website are not the fancy and famous ones which can be easily found on Yelp, but just some small-scale but special local restaurants. That is why I do not use APIs.

I used to plan to use MySQL database to store and retrieve data. But I encountered a problem that my application was not able to retrieve data stored in MySQL. Fortunately, MindMeld provides a *Question Answerer module* offering powerful capabilities for creating a knowledge base (working as a database). Details will be discussed in the section *Knowledge Base* below.

3 Approach

Given this system is developed on the MindMeld Conversational AI platform, the platform provides a Step-by-Step Guide outlining the methodology used to design a dialogue system with MindMeld.

- Select the right use case
- Script your ideal dialogue interactions
- Define the domain, intent, entity, and role hierarchy
- Define the dialogue state handlers
- Create the question answerer knowledge base
- Generate representative training data
- Train the natural language processing classifiers
- Implement the language parser
- Optimize question answering performance
- Deploy trained models to production

Obviously, the use case is the Blue Ridge Parkway tour assistant. And I also show a script example in section 1. The remaining steps will be discussed further below.

3.1 NLP Model Hierarchy

An explicit architecture and hierarchy of the conversational agent can help the NLP component to better understand natural language. The hierarchy of the tour assistant resembles Fig. 1.

Three layers, *domains*, *intents*, and *entities* are used to show the hierarchy. There are seven main domains and one domain (*attractions*) to be built. In the *greeting* domain, there are five intents dealing with some basic queries like ’Hello’, ’How are you’, ’Thank you’, ’Goodbye’, and etc. For the *hiking* domain, there are three intents. In practice however, the *ask_trail_info* intent are divided into three specific intents, *ask_trail_length*, *ask_trail_milepost*, and *ask_trail_skill_level*. The entities *length*, *skill_level*, and *trail_name* indicates what entities will be used in each intent. The *faq* domain includes three intents about some frequently asked questions like detour information, speed limit, and etc. And it is really easy to be scaled to handle more and more FAQs. And the *unknown* domain is used to appropriately answer users’ questions beyond the capabilities of this system.

Some other intents of domains like *lodgings*, *overlooks*, and *restaurants* are not shown in the figure but are very likely to the *hiking* domain, include intents like *ask_overlook_elevation*, *ask_restaurant_milepost*, *recommend_lodgings*, and etc.

3.2 State Handlers

Conversational interactions consist of steps called *dialogue states*. For each dialogue state, a particular form of response is appropriate, and, particular logic may be invoked to determine certain parts of the content of

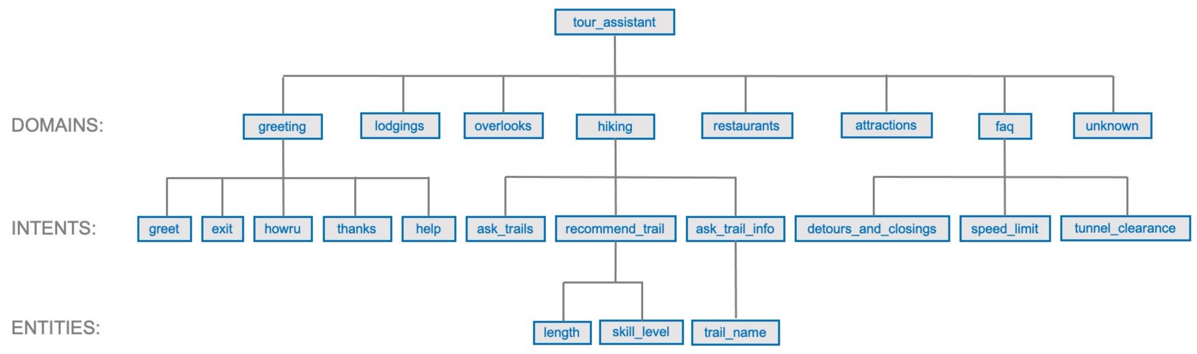

Figure 1: Design of the NLP model hierarchy

the response. A set of *dialogue state handlers* define the logic and response required for every dialogue state that a given application supports.[MindMeld(a)]

A dialogue manager is at the core of every conversational application. The dialogue manager analyzes each incoming request and assigns it to a dialogue state handler which then executes the required logic and returns a response. Fortunately, MindMeld has a powerful dialogue manager which provides advanced capabilities for dialogue state tracking, beginning with a flexible syntax for defining rules and patterns for mapping requests to dialogue states.

For example, when the system receives a query *'Hello'* , the intent classifier trained with the training data will classify it to the *greet* intent, and then invokes the following Python function and replies *'Hello'* back to the user. The *@app.handle()* decorator specifies the pattern which must be matched to invoke the handler method. And every dialogue handler uses the *responder* object to specify the natural language text and any other data to be returned in the response, and the *request* object, which stores all data passed in by the client to the application in the form of a dictionary attribute.

```
@app.handle(intent='greet')
def welcome(request, responder):
    responder.reply('Hello.')
```

With the basic understanding of the state handlers, it is easy to define the ones used in this system. When the user asks for the trail-related information like milepost, length, and skill level, a corresponding function will be invoked to retrieve data from the knowledge base, which will be discussed further below, and then reply the information back to the user.

For some more complicated functions such as *trail recommendation*, we first utilize the *entity recognizer* to figure out whether the trail length and skill level are included in the query. If so, retrieve the trails meeting these conditions in the knowledge base, otherwise, prompt questions like *'Which skill level do you prefer?'* or *'Please specify the length.'* to ask for the required information. Some other recommendation functions are similar to this one.

3.3 Knowledge Base

In order to answer users' questions about the Blue Ridge Parkway, a database or knowledge base is required to store the parkway-related information like trails, attractions, restaurants, etc.

MindMeld provides a *Question Answerer module* offering powerful capabilities for creating a knowledge base in order to demonstrate intelligent behavior in the application. The knowledge base can contain one or more indexes. Each index holds a collection of objects of the same type.[MindMeld(b)]

In this tour assistant system, there are four indexes, *lodgings*, *overlooks*, *restaurants*, and *trails*. For example, the *trails* index contains a collection of trail IDs, names, lengths, and skill levels and the *overlooks* index contains a collection of overlook IDs, names, elevation, brief introductions and links to the interactive 360degree view of overlooks. Each index is built using data from one or more JSON files. As shown in the code block below, it is the format of the *trails* knowledge base.

```
[
  {
    "id": "1",
    "milepost": 5.9,
    "region": "ridge",
    "trail_name": "Mountain Farm
       Trail",
    "length": 0.25,
    "skill_level": "easy"
  },
  {
    "id": "2",
    "milepost": 6,
    "region": "ridge",
    "trail_name": "Appalachian Trail"
       ,
    "length": 2,
    "skill_level": "strenuous"
  }
]
```

3.4 Generate Training Data

In MindMeld, there are five core supervised learning components relying on training data, *Domain Classifier*, *Intent Classifier*, *Entity Recognizer*, *Role Classifier*, and *Entity Resolver*. Furthermore, there are two different types of data files used for training in MindMeld, *Labeled Queries* and *Entity Mappings*.

Labeled query files are text files containing example user queries. I list some labeled queries in the *recommend_trails* intent below, which indicates both the label method and a labeled queries example. In these examples, I labeled entities like *easy* and *moderate* with the label *skill_level*, and labeled each number with two labels *sys_number* and *trail_length*. *sys_number* is a MindMeld built-in entity used to recognize numbers like *4*(integer), *123.4*(float), *eight*(English words), and etc.

- are there any {easy|skill_level} trails?
- show me some {moderate|skill_level} trails
- could u please recommend any {moderate|skill_level} trails?
- recommend me some {easy|skill_level} trails around {3|sys_number|trail_length} miles
- I prefer a {moderate|skill_level} trail about {1.5|sys_number|trail_length} miles

As shown in the code block below, entity mappings are JSON files which associate whitelisted alternate names, or synonyms, with individual entities. It is useful to handle the misspelling in users' queries. In this example, we can tell that *cname* is the original name of the entity, *whitelist* is a list including all the alternate names might be used in the query.

```
{
  "entities": [
    {
      "id": "1",
      "cname": "Mountain Farm Trail",
      "whitelist": ["the Mountain
          Farm Trail", "Mountain Farm
          ", "the Mountain Farm"]
    },
    {
      "id": "2",
      "cname": "Appalachian Trail",
      "whitelist": ["the Appalachian
          Trail", "Appalachian", "the
           Appalachian"]
    }
  ]
}
```

3.5 Train the NLP Classifiers

MindMeld provides a Natural Language Processor (NLP) which is tasked with understanding the user's natural language input and applies four layers of classifiers in the following order:

1. **Domain Classifier** classifies input into one of a predefined set of conversational domains. Only necessary for apps that handle conversations across varied topics, each with its own specialized vocabulary.

2. **Intent Classifiers** determine what the user is trying to accomplish by assigning each input to one of the intents defined for your application.

3. **Entity Recognizers** extract the words and phrases, or entities, that are required to fulfill the user's end goal.

4. **Role Classifiers** assign a differentiating label, called a role, to the extracted entities. This level of categorization is only necessary where an entity of a particular type can have multiple meanings depending on the context.

For developers, it is very easy to train the NLP classifiers using the command:

```
python −m <app−name> build
```

3.6 Other Steps

There are still some other steps need to be completed to optimize the system performance, like the *Language Parser configuration* and the *Question Answer performance optimization*, which will be two parts of the future work. You can check the details of these two in the MindMeld documentation.

4 Experiment & Results

To better train the NLP classifiers, the system requires a large amount of training data. The training data are labeled queries which indicate what questions will users ask the system. As a non-native English speaker, I felt struggling to generate data, to imagine how will the users interact with the dialogue system. So I tried to find out whether the data volume has an impact on training the NLP classifier (entity recognizer).

INTENT	DATA NUM	ERA
build_order	1037	0.6654
build_order	550	0.6102
build_order	100	0.4843

Table 2: The entity recognition accuracy (ERA) for the *food ordering* system.

Experiment Setup Due to the insufficient data of this tour assistant system, I did the experiment with another MindMeld based *food ordering* system. It includes both the training data and test data for each intent. But only the *build_order* will be considered, because it is the only intent involving the entity recognition task. The system will train the classifier (entity recognizer) with the full training data for the first time, and with the incomplete training data (deleted half of the data randomly) for the second time. Then use the same test data to evaluate them. The *entity recognition accuracy* will the metrics. And the results are shown in Table 2.

Results As shown in Table 2, I evaluated the entity recognizer with the same test data. More data I feed into the system, higher results I will get. In this experiment, the smallest dataset I use includes 100 queries, but the largest dataset I use in the tour assistant system contains less than 40 queries. Therefore, you can imagine that there is a huge space for this system to improve.

5 Evaluation

Two metrics are used to evaluate the system. One is the *objective metrics* which used the MindMeld built-in tool to evaluate the system automatically and the other one is the *subjective metrics* which asks real users to judge it.

DOMAIN	ICA
greeting	0.9775
faq	1.0
hiking	1.0
overlooks	0.9643
restaurants	1.0
lodgings	0.9638
unknown	0.9750

Table 3: The intent classification accuracy (ICA) for the *tour assistant* system.

5.1 Objective Metrics

MindMeld provides a built-in evaluation tool which can easily get the *domain classification accuracy*, the *intent classification accuracy*, the *entity recognition accuracy*, and the *role classification accuracy* for each domain in the application. Obviously, these metrics indicate how well the classifier performs for different domains and intents. The higher accuracy we get, the better the system performs.

However, I only generated the test data for the domain and intent classification evaluations. Therefore, the entity recognition accuracy and the role classification accuracy are not available. To start the evaluation, use the following command:

```
python -m <app-name> evaluate
```

Then MindMeld will automatically evaluate the classifiers. For the seven domains in the system, the system's domain classification accuracy is *0.9821* which is pretty high. And the intents classification accuracy for all the intents in each domain are shown in Table 3.

As shown in Table 3, the intent classification accuracy for all the domains are very high, some of them even reached 1.0. But as I mentioned in the *Experiment & Results* section, both the training and test data I generated are very limited and even very close to each other. That is probably why the result looks so perfect. Therefore, one of the following work is to find more users to evaluate it and help me generate more training and test data (labeled queries).

5.2 Subjective Metrics

As a functional application, the user experience is always a top priority, therefore I designed a questionnaire for it, ask some real users to interact with it, then fill the questionnaire, and give a overall rating of the interaction from 1 to 5. Some question examples are shown below:

- *Did you complete the conversation successfully?*
- *Did the system understand what you said?*
- *Was it easy to find the information you want?*
- *Was the pace of interaction with the system appropriate?*
- *Did the system answer all your question with appropriate answers?*
- *If not, what questions the system failed to answer?*
- *Did you know what you could say at each point of the dialog?*
- *Do you think you'd use the system regularly in the future?*
- *Did you encounter any other problems with the system?*

Since this system has not yet been released, I did not find too many users to test the system. All I got is the feedback from 9 of my friends. The feedback will be discussed in the *Discussion* section.

5.3 Other Metrics

I think the metrics mentioned above work for this functional system, but I still want to use these four metrics below to better evaluate it. However, the MindMeld built-in evaluation tool does not support them. I am still trying to find some other ways to implement them, and this will be a part of the future work.

- number of turns or utterances

- dialogue time or task completion time
- mean user response time
- mean system response time

6 Discussion

After the evaluation, 2 of the 9 users gave 5 points (out of 5), 4 of them gave 4 points, and 3 of them just gave 3 points, which met my expectation. But there are still some problems.

A frequently problem mentioned by the users in the questionnaire is about the *lodgings* and *restaurants* domains. After the system recommending some hotels or restaurants, users said they could not figure out the price of them, which is a really important attribute for them to decide whether they want to go there. Although a website for each restaurant or hotel is available is the knowledge base, users barely ask questions like *'Show me the website of the restaurant'*. To solve this problem, I plan to add more intents like *ask_lodging_price*, *ask_restaurant_menu*, and etc. When these intents are recognized by the classifier, instead of answer the questions directly, the link of the website and a proper prompt will be sent to users, because it is not a Yelp like system after all.

Some users also mentioned that they did not know what they could say at some points of the dialogue. Sometimes there is a prompt, sometimes there is not. This might be an oversight of my design. I will check the codes and do the appropriate modification.

7 Conclusion

If this project succeeds, as a utility tour assistant, we can put it on the Blue Ridge Parkway official website or integrate it to the *Blue Ridge Pkwy Travel Planner* application to benefit society. For the users who want to quickly get the information they want, they can simply click the chat button somewhere on the web page or in the app and ask their questions. Then the conversational agent will reply to their questions with texts directly, or with an URL, which can redirect visitors to the corresponding web page.

In order to reach this great goal, there is still a long way to go. This system is now only able to answer limited questions related to trails, lodgings, restaurants, overlooks, and FAQs. But the Parkway also has a lot of niche activities worth exploring for tourists, besides mainstream activities like hiking, camping, etc. For example, people can find special country music played by locals, wineries offering premier wine experience, etc. Therefore, we need to perfect the data, and enrich the system structure to deal with different domains.

References

[Foundation()] The Cultural Landscape Foundation. Blue ridge parkway. `https://tclf.org/landscapes/blue-ridge-parkway`.

[MindMeld(a)] MindMeld. a. Step 4: Define the dialogue state handlers. `https://www.mindmeld.com/docs/quickstart/04_define_the_dialogue_handlers.html`.

[MindMeld(b)] MindMeld. b. Step 5: Create the knowledge base. `https://www.mindmeld.com/docs/quickstart/05_create_the_knowledge_base.html`.