Human Machine Dialog Project

Mohamed Mazhar (241789)

University of Trento

m.mazhar@studenti.unitn.it

1. Introduction

In this project, I created a virtual assistant capable of helping tourists find various facilities, such as restaurants, in a city. I chose to undertake this type of project because, based on my personal experience, visiting an unfamiliar city can be very challenging. Navigating and finding all the necessary information easily and quickly is often complicated. This virtual assistant, which I named CityGuideBot, aims to solve these problems by providing precise and timely information about the various services available in the city.

CityGuideBot can help tourists find restaurants based on location, type of cuisine, and price range. In addition to restaurants, the assistant provides information on local events, tourist attractions, hospitals, and other tourist facilities. Through simple voice conversations, CityGuideBot can manage restaurant bookings, indicate the nearest emergency room in case of an emergency, and offer assistance in critical situations, such as sending an ambulance.

This project not only makes life easier for tourists but also contributes to improving the tourism industry of a city. A virtual assistant like CityGuideBot can enhance the overall visitor experience, making their stay more enjoyable and less stressful. Ultimately, such a tool can attract more tourists, thereby boosting the local economy.

2. Conversation Design

The virtual assistant for tourism was created with three key principles: being useful, easy to use, and proactive. The assistant needs to handle different types of conversations based on common situations in the tourism sector. Therefore, the assistant is not designed for casual chat; common questions like "how are you?" are directed to a specific help intent.

2.1. General Lookup Scheme: Restaurants and Events

Since my assistant is designed to work in a tourism context, I provided it with a specific list of restaurants and events stored in a JSON database. This list can be updated as new places and activities come up and old ones are replaced.

The assistant is trained to search for restaurants and events based on specific characteristics such as price range and location. For simplicity, we use three location categories: city center, Trento Nord, and Trento Sud.

When a user wants to find a restaurant or event, the assistant first asks for the relevant details, such as the desired location and price range. It then takes these inputs and searches the JSON database for matches.

If there are multiple matches that fit the criteria, the assistant presents the best options to the user, asking them to select one of the suggested places. For example, the user might be asked to choose a restaurant based on the given characteristics and, if needed, refine their search until the best match is found. This process is managed via a Rasa form: the assistant will keep requesting more details until the user picks one of the suggested options or terminates the conversation.

This approach ensures that the assistant can provide accurate and relevant information based on the user's preferences and the current list of restaurants and events available in the database.

2.2. Conversation

The CityGuideBot is designed to handle a variety of conversations tailored to the specific needs of users in the tourism sector. These conversations are modeled to assist users in finding and booking restaurants and events, managing their reservations, and obtaining information about the available options. Below is a detailed description of the different types of conversations you can have with the assistant.

2.2.1. Finding a Restaurant

When a user wants to find a restaurant, the conversation begins with the assistant asking about the user's preferences. The assistant will inquire about the desired location (city center, Trento Nord, or Trento Sud) and the type of cuisine the user is interested in. For instance, the user might say, "I want to find an Italian restaurant in Trento Nord." The assistant will then ask for additional details such as the preferred price range (e.g., moderate, high-end).

Based on the user's inputs, the assistant searches the JSON database for restaurants that match the specified criteria. If multiple matches are found, the assistant will present the best options to the user, such as "I found three Italian restaurants in Trento Nord with moderate prices. Would you like to book one?" The user can then select one of the options. Once the user makes a selection, the assistant can provide further information, such as the restaurant's address, opening hours, and contact details.

2.2.2. Booking a Restaurant

After finding a suitable restaurant, the conversation can smoothly transition into the booking process. The assistant will ask the user if they would like to make a reservation. If the user agrees, the assistant will proceed by asking for the necessary booking details: date, time, and number of people. For example, the assistant might say, "Great! When would you like to book a table? And for how many people?"

The user provides the date such as "30/06/2024" and the time, such as "7:00 PM" and the number of people, like " 2 ." The assistant then confirms the booking with a summary: "You have booked a table for two at La Dolce Vita on June 30, 2024, at 7:00 PM. Is that correct?" Upon confirmation from the user, the assistant finalizes the reservation and provides a booking confirmation message.

2.2.3. Finding an Event

The assistant also helps users find events based on their interests and location preferences. The conversation starts with the user asking for events in a specific area. For example, the user might say, "Are there any events today?"

The assistant will then ask for more specifics, such as the location or any particular preferences for the event. Using these details, the assistant searches the database for matching events and presents the options to the user.

2.2.4. Managing Reservations

Users can also manage their existing reservations through the assistant. The conversation starts with the user expressing their intent, such as "I want to see my reservations" or "I need to cancel a booking." The assistant then retrieves the user's current reservations from the database and presents them with the options.

For viewing reservations, the assistant lists all active bookings, such as "You have a table reserved at La Dolce Vita on June 30, 2024."

If the user wants to cancel a booking, the assistant will ask for the specific reservation they wish to cancel. For example, "Which reservation would you like to cancel? The table at La Dolce Vita?" Once the user specifies, the assistant confirms the cancellation and updates the database, providing a cancellation confirmation message to the user.

2.3. User Experience and Flow

Throughout these conversations, the assistant maintains a user-friendly and proactive approach. It guides the user step-by-step, ensuring clarity and ease of use. The assistant's design aims to make the interaction as smooth and efficient as possible, reducing the need for the user to repeat information and providing timely confirmations and summaries.

In all scenarios, the assistant balances between guiding the user and allowing them the freedom to make choices. This interaction model ensures that users feel supported and in control, enhancing their overall experience with the virtual assistant.

By managing these diverse types of conversations, the virtual assistant effectively addresses the various needs of users in the tourism sector, making it a valuable tool for planning and enjoying their travel experiences.

3. Data Description & Analysis

3.1. Dialogue train data

CityGuideBot was trained on different types of conversations, which I collected in a file called `stories.yml`. This file contains two types of stories. The first type was entirely created by me, while the second type included the help of some friends who tested the assistant.

The `stories.yml` file contains all the conversation parts that can be addressed during a dialogue with the assistant. In total, I created 20 stories. I decided to create more shorter stories rather than a few complex ones, as this led to better results. By increasing the recombination of dialogues, the assistant is able to be more flexible with respect to any changes in the direction of the dialogue flow.

I also had to take into account different case histories, such as the various choices when selecting restaurants or events. This led to an increased number of story fragments to be created. To manage the various possible dialogue flows, I found the Rasa-provided checkpoints very helpful, as they allowed me to produce cleaner stories and consider different conversational paths that might arise.

3.2. NLU train data

CityGuideBot's Natural Language Understanding (NLU) component was trained on various intents and entities, collected in a file called `nlu.yml`. This file contains examples for different intents that the assistant can recognize and respond to.

In my project, I collected 25 possible user intents and defined them textually within the `nlu.yml` file. The total number of samples is 155, averaging 6.2 samples per intent. I crafted these samples manually to cover the majority of possible expressions used by a user to define a precise intent.

Within these intents, it was necessary to train the model to recognize entities, which I used to create the logical structure of the assistant. For example, the model can recognize entities like `location`, `cuisine`, `people`, and `date`, allowing the assistant to understand and respond to specific details provided by the user.

By providing a diverse set of examples for each intent, CityGuideBot is able to understand and respond accurately to a wide range of user inputs. This comprehensive dataset helps improve the flexibility and reliability of the assistant in real-world interactions.

3.3. Domain data

In my project, I defined the domain of CityGuideBot in the `domain.yml` file. This file includes intents, entities, slots, responses, and actions that the assistant can handle.

3.3.1. Intents

The domain file specifies 22 user intents that CityGuideBot can recognize, such as `greet`, `find_restaurant`, `inform_location`, `report_emergency`, and `cancel_booking`.

3.3.2. Entities

Entities are used to extract specific information from user inputs. The domain file includes entities like `people`, `location`, `restaurant`, `cuisine`, `price_range`, `date`, `time`, `rating`, `filtered_restaurants`, and `booking`.

3.3.3. Slots

Slots store extracted information from user inputs. Each slot is defined with a type and mappings to corresponding entities. For example, the `cuisine` slot stores the type of cuisine the user is looking for, and it maps to the `cuisine` entity.

3.3.4. Responses

Predefined responses are used by the assistant to interact with users. The domain file contains various responses, such as:

- `utter_greet`: "Hello! How can I assist you today?"
- `utter_goodbye`: "Goodbye! Have a great day!"
- `utter_thanks`: "You're welcome!"
- `utter_ask_cuisine`: "What type of cuisine are you looking for?"

- `utter_confirm_booking`: "I have booked a table for {people} people at a {cuisine} restaurant with {rating} rating in {location} on {date} at {time}. Enjoy your meal!"

3.3.5. Actions

Custom actions perform specific tasks based on user inputs. Examples of actions in the domain file include:

- `action_filter_events`
- `action_filter_restaurants`
- `action_book_table`
- `action_show_bookings`
- `action_cancel_booking`

By structuring the domain in this way, CityGuideBot can effectively understand and respond to user queries, manage bookings, provide information about restaurants and events, and handle emergencies.

4. Conversation Model

4.1. Model, algorithm, tokenizer, featurizer and extractor

In developing CityGuideBot, I tested two different pipelines to process and analyze user inputs effectively.

4.1.1. Pipeline 1

First, I set up a pipeline that uses the **WhitespaceTokenizer** to split the input text based on spaces. This helped break down user input into smaller parts for easier processing. Then, I used the **CountVectorsFeaturizer** to turn the text into a matrix of token counts, which is a basic step for text processing tasks. I also included another **CountVectorsFeaturizer** with the **char_wb analyzer** to create features from character n-grams (1 to 4 characters). This helps the model understand different forms of words by looking at parts of words like prefixes and suffixes.

For classifying user intents, I used the **LogisticRegressionClassifier**, which uses logistic regression to categorize intents based on the feature vectors. This method is simple and efficient, making it suitable for reliable intent detection. To extract entities from the text, I used the **CRFEntityExtractor**, which uses Conditional Random Fields (CRF) to accurately find and extract entities. Although I considered using the **DucklingEntityExtractor** for extracting things like numbers and amounts of money, I decided not to use it for this implementation. Finally, I included the **ResponseSelector**, which I trained for 2 epochs. This component selects the best response from predefined responses based on the user's input and the context of the conversation, ensuring the responses are relevant and improve the user experience.

4.1.2. Pipeline 2

In the second pipeline, I used **SpacyNLP** with the **en_core_web_md model** for natural language processing tasks. This provided advanced language understanding capabilities. Next, I used the **SpacyTokenizer** to split the text into individual tokens using Spacy's method. For creating feature representations of the text, I used the **SpacyFeaturizer**, which converts tokens into feature vectors. I also included the **RegexFeaturizer** to extract features based on regular expressions and the **LexicalSyntacticFeaturizer** to add lexical and syntactic features to the tokens.

For intent classification, I used the **DIETClassifier**, a dual intent and entity transformer classifier, which I trained for 50 epochs. To ensure consistency in recognizing entities, I used the **EntitySynonymMapper**, which maps synonyms to the same entity. Finally, I used the **ResponseSelector** to choose the most appropriate response from predefined responses, also trained for 50 epochs. This comprehensive pipeline ensures that CityGuideBot can accurately understand and respond to various user queries and commands.

4.2. Response Policy & Fallback Policy

To manage the dialogue flow and ensure accurate responses, I configured several policies in CityGuideBot. These policies help in deciding the next action based on the user's input and the current state of the conversation.

4.2.1. TEDPolicy

First, I used the **Transformer Embedding Dialogue (TED) policy**, which employs a transformer model to predict the next action. This policy considers the context of the conversation up to 7 previous turns and was trained for 10 epochs. The TEDPolicy is instrumental in handling complex and context-dependent interactions by leveraging the power of transformers.

4.2.2. AugmentedMemoizationPolicy

Additionally, I implemented the **AugmentedMemoizationPolicy**, which can generalize across similar conversations. This policy extends the memoization capabilities, making it more flexible with respect to slight variations in dialogue. It remembers the exact conversations it has seen during training and can predict the same next action if a similar conversation occurs, handling frequently occurring dialogue patterns efficiently.

4.2.3. RulePolicy

The **RulePolicy** was another key component in my configuration. It allows me to define specific rules that override other policies in certain situations. For instance, the bot follows predefined rules for handling fallback scenarios or specific user intents that require a consistent response. This ensures that the assistant behaves predictably in critical situations.

4.2.4. UnexpecTEDIntentPolicy

Lastly, I used the **UnexpecTEDIntentPolicy** to handle cases where the bot's confidence in its prediction is low. This policy triggers fallback actions, such as asking the user to rephrase their input or providing a default response. It was trained for 10 epochs and includes settings like an NLU threshold of 0.3 and an ambiguity threshold of 0.1. The policy enables fallback predictions with a core fallback threshold of 0.4 and ensures fallback actions are performed when needed by setting the fallback action name to **action_default_fallback**.

These policies together ensure that CityGuideBot can handle a wide range of conversational scenarios, providing accurate and contextually appropriate responses to user inputs.

5. Results

6. Evaluation of CityGuideBot Performance

The performance of CityGuideBot was evaluated using two different pipelines. The results of these evaluations are presented below.

6.1. Pipeline 1

- F1-Score: 0.846
- Precision: 0.861
- Accuracy: 0.842

Pipeline 1 achieved an F1-Score of 0.846, which indicates a balanced performance between precision and recall. The precision of 0.861 suggests that the model is very accurate in its positive predictions, meaning it correctly identifies the intended actions with a high degree of confidence. The overall accuracy of 0.842 shows that the model correctly predicts the user's intent and entities in 84.2% of the cases. While these results are strong, there is room for improvement in achieving higher accuracy and precision.

6.2. Pipeline 2

- F1-Score: 0.922
- Precision: 0.925
- Accuracy: 0.926

Pipeline 2, on the other hand, performed better across all metrics. It achieved an impressive F1-Score of 0.922, indicating a very balanced and effective performance. The precision of 0.925 demonstrates that the model's positive predictions are highly accurate, even more so than Pipeline 1. The overall accuracy of 0.926 shows that the model correctly predicts the user's intent and entities in 92.6% of the cases, which is a significant improvement over Pipeline 1.

6.3. Comparison and Analysis

Comparing the two pipelines, it is evident that Pipeline 2 outperforms Pipeline 1 in all evaluated metrics. The higher F1-Score, precision, and accuracy indicate that Pipeline 2 is more reliable and efficient in understanding and responding to user inputs. This suggests that the use of SpacyNLP, advanced featurizers, and the DIETClassifier in Pipeline 2 provides a better understanding of the natural language, resulting in more accurate predictions.

Overall, while both pipelines show strong performance, Pipeline 2's enhanced metrics suggest it is better suited for the tasks handled by CityGuideBot, offering more accurate and reliable dialogue management.

7. Conclusion

According to the people who tried CityGuideBot, the idea of a tourism bot is valid, simple to use, and very useful. For future work, it would be interesting to put it on Instagram and other social media platforms so that young people can use it more. Another future goal is to propose it to various cities to help tourists find information about the city's life.