Report SHRI

Matteo Germano id:1807599, Federica Cocci id:1802435

January 2021

Contents

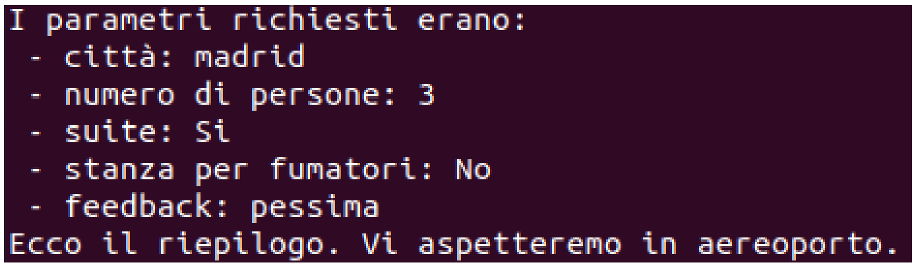
1 Abstract

The goal of the project is the implementation of a simple spoken italian chatbot. We have chosen to use the Rasa framework to handle the bot implementation and Google Speech Recognition with pyttsx3 to handle the voice interaction between user and bot.

We have implemented an assistant for hotels research in european capitals. In particular it is a formbot which stores the user information and at the end it returns a recap. These informations are: the hotel city, the number of people, if the user wants a suite or a simple room, if the user wants a smoker bedroom, eventually a feedback of the user about the experience of interaction with the bot and finally if the user wants to keep hotel crew waiting for him at the airport when he arrives in the city.

2 Rasa

Rasa is an open source machine learning framework for automated text and voice-based conversations. Understand messages, hold conversations, and connect to messaging channels and APIs.



```
I parametri richiesti erano:
 - città: madrid
 - numero di persone: 3
 - suite: Si
 - stanza per fumatori: No
 - feedback: pessima
Ecco il riepilogo. Vi aspetteremo in aereoporto.
```

Figure 1: example of Recap.

2.1 NLU

The goal of NLU (Natural Language Understanding) is to extract structured information from user messages. This usually includes the user's intent and any entities their message contains. We used nine types of intent.

The most important are:

- Request_hotel: it contains examples of messages that the user can send to the bot in order to request for an hotel.
- Inform: it contains examples about city requestes, feedback, number of people and type of room.
- Affirm: affermative types of answers.
- Deny: negative types of answers.
- Greet: examples of greetings to start a conversation.

We also used entities, that are structured pieces of information inside a user message. In particolar our entities are: city, room, number, feedback, airport. We used Duckling to parse text and entities into structured data.

2.2 Stories

Stories are a type of training data used to train our assistant's dialogue management model. Stories can be used to train models that are able to generalize to unseen conversation paths.

We have written two stories:

- Stop form and continue: It starts with the activation of the form and, if the user wants to stop, the bot asks for going on with the compilation of the form and the user affirms, so the bot will continue with the form.
- Stop form and stop: It starts with the activation of the form and, if the user wants to stop, the bot asks for going on with the compilation of the form and the user denies, so the bot will stop with the form.

2.3 Rules

Rules are a type of training data used to train our assistant's dialogue management model. Rules describe short pieces of conversations that should always follow the same path.

We have written some rules but the most important ones are:

- Activate hotel form: The user requests for an hotel and the bot activates the form.

- Submit form: if the form is active and every value is set, the bot submits the form.

2.4 Domain

The domain defines the universe in which our assistant operates. It specifies the intents, entities, slots, responses, forms, and actions our bot should know about. It also defines a configuration for conversation sessions.

In the domain file we have listed entities, intents, slots (our bot's memory. They act as a key-value store which can be used to store information the user provided as well as information gathered about the outside world), responses, actions and forms. The only form we use is linked with slots and actions: the form is composed by slots and is validated by the actions. For the responses we only wrote one type of message for every response, but it is possible to wrote many of these and the bot will choose randomly one among them.

2.5 Actions

The actions are written in a python file. It could be seen as the living soul of the bot. For some slots values is required a validation such as the number of people and the hotel city. For cities we use an internal city database (with every capital in Europe) and the validation check if the city required by user is actually in the database; if it is false the bot will inform the user that the service is not available in that city.

3 VoiceCommand.py

In this python file we used the Google Speech Recognition, pyttsx3 and some util libraries such as rasa.core and rasa.utils.

- Google Speech Recognition: they are Google APIs that we use to have a textual representation of a spoken message.

- pyttx3: a python library to convert written text to a spoken message. It can be used in different human languages. For this project we have chosen the italian language.

- rasa.core: to create a link between the spoken part and the rasa implementation of the bot.
- rasa.util: to have a link with the action server.

There is a main loop where the first thing is the voice recognition and its conversion to text. Then the input, thanks to the agent imported from rasa.core, gets a textual answer from the bot that will be repeated out loud. If the user says the word "stop", the main loop ends and the bot also stops.

4 Conclusions

A chatbot for a real use is trained with a lot of examples: if we were building a real chatbot, we would use a Conversation-Driven Development approach to increase the set of intents in NLU to improve the overall accuracy and the bot performance. Conversation-Driven Development is the process of listening to users and using those insights to improve the AI assistant.

Even if the results are not always perfect, it is acceptable for our bot.