Language Understanding Systems
Final Project

Riccardo SCILLA - 213723
riccardo.scilla@studenti.unitn.it

Abstract

The aim of this project is to implement a contextual assistants chatbot using Rasa and to connect it to Alexa as custom skill to get a voice-enable assistant.

1 Introduction

The Alexa skill built in this project is a Bartender Bot whose aim is to explain an suggest different types of cocktails and drinks depending on the dialogue with the user. The skill can execute two main operations:

1. Search and suggest cocktails based on a list of ingredients
2. Briefly explain each cocktail and its preparation

These operations can be executed independently or one after the other following the flow of the conversation.

2 System Description

2.1 Dataset

The dataset used for the implementation of this skill is taken from *theCocktailDB*, making some api calls in order to create a small dataset in a xlsx format, using a python script. The final dataset contains 65 cocktails. The values of the cocktails extracted from the database are: name, category, IBA category, if alcoholic, type of glass, ingredients with measures, instructions and a link to an image.

2.2 NLU

The NLU file containing the intents is created using *chatette*, a Python program that generates training datasets for Rasa NLU given a template files.

2.3 Search task

The search task has the goal to search and suggest desired a cocktail. To properly execute this task, a *search_form* is built and user must provide the category of the cocktail and at least one ingredient. If something is missing the bot responds with a failure message.

In *search_form*, the bot will answer adaptively depending on the different category and ingredients choose so far:

Category

If not specified, the category is asked as first slot. The input is checked in the database and if not present, the bot suggests which ones are the possible categories.

Ingredients

If not specified, the ingredient is asked as second slot. The user could ask for multiple ingredients in a single utterance: each one is checked in the database and if not present, the bot responds with a failure message. In positive case, the ingredient is inserted in a *ingr_list* slot which contains all the valid ingredients chosen.

In some cases, an ingredient name is present in multiple ingredients (eg. *"rum", "white rum", "dark rum"*). In these situation a *multiple_options* slot flag is set to *true* and the system performs **disentanglement**: it lists all possible matched ingredients and asks the user to choose one of those.

The system will repeatedly ask for ingredients. To give a more natural feeling, the requests are different if the *ingr_list* slot is empty or not (ie. *"Which ingredient do you want first?" "Which ingredient do you want to add?"*) and the bot affirms when the ingredients inserted are valid, to disambiguate the fact the it understood or not what the user asked. In this part of the conversation, a *ask_ingredients* intent can be inserted by the user to have have all chosen ingredients listed in one single message.

The *search_form* stops when the user specifies a *stop* intent.

At the end of the form, the system will ask for a final **confirmation** by the user with an *action_ask_confirm*, listing category and ingredients slots: if affirmed, the conversation continues. The system performs a *search_action* checking in the database for the cocktails that match the category and the ingredients: every matched cocktail is inserted in a *proposed* slot. Here the user can face multiple situations:

1. No proposes: the system returns a failure message.

2. One propose: the system proposes it.

3. More proposes: the system enumerates cocktails in pages of three items per times

If the items are more than three, the bot asks if the user wants to see more results and continues in this way until a *stop* intent is recognized.

In the case of more proposes, a *choice_form* is executed and the user must provide a cardinal value from the enumerated list showed before. If the value is not valid, the system returns a failure message. At this point, an affirmation utterance is displayed and the flows continues with the **Explanation task**.

2.4 Explanation task

The explanation task has the goal to provide information about a specific cocktail. This task can be executed independently or immediately after the **Search task**, with reference to the cocktail chosen in that part.

1. If ran independently, the user must provide the name of cocktail.

2. If ran following the Search task, the system asks if the user wants to see more information about the cocktail just chosen.

When the user asks for an explanation about a cocktail, the system will tell if the cocktail is alcoholic and its complete list of ingredients.

The user can asks for a detailed preparation: in this case, the system provides the measures of each ingredient and a short text with instructions the steps on how to prepare it.

2.5 Domain

In the domain file are specified the intents, entities, slot and actions that the bot knows about. The different components are briefly described in Table 1.

3 Configuration

The aim of this part of the project is to try different NLU and Core configurations in order to achieve the best possible results.

	Description
Intents	
greet	greet the bot
goodbye	leave the bot
thanks	thank the bot
affirm	affirmative response to bot's question
deny	negative response to bot's question
ask_functions	ask the bot on his capabilities
ask_ingredients	ask to show the ingredients chose so far
search_provider	specify category and/or ingredient(s)
explain	ask for explanation of a specific cocktail
ask_preparation	ask for preparation of a specific cocktail
ask_repeat	ask to repeat
cardinal	specify a cardinal value
Entities	
category	text, category of the searched cocktail
ingredient	list, list of ingredients inserted by the user
cocktail	text, cocktail chosen by the user or found with *action_search*
ingr_list	text, list of valid ingredients
multiple_options	bool, true if there are many ingredients containing the name inserted by the user
proposes	text, list of cocktails that match category and ingredients
more_results	bool, true if there are other more three proposes to show
cardinal	text, value in the list of proposed cocktails
Actions	
action_ask_confirm	to confirm the category and ingredients in *search_form*
action_list_ingredients	to list the ingredients chosen so far in in *search_form*
action_search	to search the cocktails matching with category and ingredients
action_explain	to explain the specified cocktail
action_preparation	to show preparation steps of the specified cocktail
action_repeat	to repeat the previous messages displayed by the bot
action_continue	to continue the flow of the conversation
Forms	
search_form	requires category and ingredients, validate them and loops until user insert a *stop_intent*
choice_form	requires cardinal and validate it

Table 1: Description of domain components

3.1 NLU Evaluation

Rasa NLU is a natural language processing tool used for intent classification, response retrieval and entity extraction in chatbots, considering NLU data described in Section 2.2. To evaluate the NLU, different pipelines have to be defined and consist of three main parts:

- Tokenization
- Featurization
- Entity Recognition / Intent Classification / Response Selectors

For each of these parts, Rasa provides many components that can be used to build our own pipeline, together with some default pipelines that can be used as first attempt.

To test the the NLU model I used both default and custom pipelines, mixed all these components in order to see how the system behaves. The results are reported in Table 2.

NLU pipeline	F1
HTF_CRF_Sklearn	0.906
Mitie_Sklearn	0.930
Mitie_DIET	0.972
Spacy_CRF_Skearn	0.944
Spacy_DIET	0.943
Whitespace_DIET	0.907

Table 2: NLU pipelines results

3.2 Core Evaluation

Rasa Core is a dialogue engine used for decide which action to perform next, considering some stories written in stories.md file. Also in this case, Rasa provides different components that can be used to build some policies. The two most important policies are Keras Policy and TED Policy.

The Keras Policy is based on an LSTM; it can be configured with *MaxHistoryTrackerFeaturizer* or *FullDialogueTrackerFeaturizer* which tell if the policy has to consider a small part of the history or the full conversation. Another parameter is the state featurizer that is used to convert the features: if *BinarySingleStateFeaturizer*, it creates a binary one-hot encoding; if *LabelTokenizerSingleStateFeaturizer* it creates a vector.

The TED Policy is based on a transformer which can be configured with many parameters such as: number of epochs, number of hidden layers, batch size and max history considered.

To test the the Core model I mixed all these parameters in order to see how the system behaves. The results are reported in Table 3.

Core policy	Correct
Keras_FD_Binary	17/20
Keras_MH5_Binary	15/20
Keras_MH5_Binary_FB	16/20
Keras_MH5_LabelTokenizer_FB	17/20
TED_MH1_E100	14/20
TED_MH1_E200	13/20
TED_MH3_E100	17/20
TED_MH5_E100_FB	17/20
TED_MH5_E100_HL2	17/20
TED_MH5_E100_HL3_FB	12/20
TED_MH5_E100_HL3	13/20
TED_MH5_E100	17/20
TED_MH5_E200_HL3_FB	16/20

Table 3: Core policies results

End-to-End		Precision	Accuracy	F1 score
NLU	Core			
Whitespace_DIET	Keras_FD_Binary	0.813	0.811	0.809
Whitespace_DIET	Keras_MH5_LabelTokenizer_FB	0.839	0.836	0.835
Whitespace_DIET	TED_MH5_E100_FB	0.853	0.841	0.843
Whitespace_DIET	TED_MH5_E100_HL2	0.845	0.834	0.836
Spacy_CRF_Sklearn	Keras_FD_Binary	0.794	0.791	0.790
Spacy_CRF_Sklearn	Keras_MH5_LabelTokenizer_FB	0.830	0.823	0.825
Spacy_CRF_Sklearn	TED_MH5_E100_FB	0.837	0.825	0.828
Spacy_CRF_Sklearn	TED_MH5_E100_HL2	0.833	0.829	0.829
Mitie_DIET	Keras_FD_Binary	0.831	0.829	0.828
Mitie_DIET	Keras_MH5_LabelTokenizer_FB	0.848	0.847	0.846
Mitie_DIET	TED_MH5_E100_FB	0.854	0.851	0.851
Mitie_DIET	TED_MH5_E100_HL2	0.862	0.861	0.860

Table 4: End-to-end results

3.3 End-to-end Evaluation

This final evaluation evaluation allow me to test dialogues end-to-end by running through test conversations and making sure that both NLU and Core make correct predictions. To do this, I collected 55 test stories in the end-to-end format: 30 written by myself, trying to generalize as much as possible the input, 25 from testers. All of them are saved in conversation_tests.md file. To test the the End-to-end model I mixed the NLU pipelines and Core policies to see how the system behaves. The results are reported in Table 4. Looking at the results we can see that all the tested models perform around the same F1 score. Slightly better results are obtained using Mitie_DIET NLU model.

4 Conclusion

The system designed is able to follow some basic paths of conversation with the user, recognizing its needs and providing information about cocktails. However it has some limitations due to the lack of variety of the train utterances. To improve the overall system behaviour, it should be distributed to some testers and the train set should be updated depending on the interactions with them. Also the stories are limited to my knowledge and other users could follow some other paths that I did not consider, therefore making it perform in an unwanted way.