A Rasa assistant to order pizza

Leonardo Lovato (221249)
University of Trento
Via Sommarive, 9, 38123 Povo, Trento TN
leonardo.lovato@studenti.unitn.it

Abstract

This paper describes the design and implementation of a conversational agent to allow users order pizzas for delivery. The system is developed with Rasa[7], a powerful open-source framework that uses state-of-the-art natural language understanding techniques.

1 Introduction

From a low level perspective, a digital assistant is an algorithm able to understand what the user is asking so as to provide a relevant answer. The proposed system is a **task-based** assistant which aims to guide the user throughout the process of ordering pizzas for delivery. Indeed, the main function allows the user to choose from which store make the order, add items (pizzas or drinks) to the shopping list, choose a delivery address, a time for delivery and set a telephone number. When all this required information are collected, a summary is shown and the user can decide to confirm, change or cancel the order. If an order has been made, functions to retrieve information about the status or to cancel it are available. Furthermore, it is possible to filter pizzas by typology (i.e. vegetarian or spicy), ask for a specific item (pizza or drink) price or seek the list of ingredients of a certain pizza.

To begin the ordering process, the user may either greet the agent to then be prompted about creating a new order or directly say his intent of ordering pizza. At this point, the model will begin the collection of necessary information in a **mixed-initiative**, **answer-question** approach (i.e. asking if something else need to be ordered after choosing some items or collecting entities given the correct intent). During the process, asking the user to show the menu of a restaurant will output the available pizzas for the chosen store, asking for the drinks will provide the orderable beverages. Moreover, asking what are the *vegetarian*, *vegan* or *spicy* options will filter the pizzas list; querying what are the ingredients of a particular pizza will return all the toppings. Lastly, an answer will be provided in case the user asks why the address, time or telephone is necessary.

Once at least one order has been made, the user may ask to get updates and the assistant will reply with the list of active orders, their details and status. In contrast, if the user changed his mind, asking to cancel the order will ask to confirm which order (if more are present) needs to be deleted to then proceed with the request.

The user target group is any person who is interested in ordering pizzas for delivery, the assistant will guide throughout the process and should be able to recover from errors or unexpected inputs, going back to the correct step of the procedure. Therefore, no special knowledge for its usage is required.

2 Conversation Design

The dialogues that the system can handle are the followings:

- **Order related:** create, modify or delete an order, add items, set address/time/telephone.
- **Information retrieval** functions of the assistant, list of ingredients, price of item, shops' menu, typologies of pizzas, status of the order.

To start an order, a pizza shop must be chosen from the available list. Thus, any user intent that triggers an action related to the ordering process will check if a shop has been chosen and prompt the shops list in the opposite case. After choosing any item, the system will acknowledge the user about the current shopping list and ask if something else is necessary. From this point, the **current_step** will be stored and used to go back to the correct action in case the user diverts from the path. This information also allows to set multiple slots in a single phrase (i.e. the store and the pizzas) and to safely add or change something without the need to set again already given information: for instance, if at the order summary the user adds another pizza, the next step will be again the summary, as all the others information were already set.

Out-of-scope, **unexpected** and **fallback** intents have been taken care following the rasa documentation[2], in particular:

- **Out-of-scope:** informing the user about the impossibility to handle the request.
- **Unexpected:** telling the user that the function is not available yet or giving explanation why that information is required (i.e. for the address, time of delivery and telephone).
- **Fallback:** asking the user to rephrase the intent.

3 Training Data & Analysis

At the earliest stage of development, all the training data has been written manually. When a first set of functions where implemented, the **rasa interactive**[4] command has been used to generate stories and NLU data. Dialogues were collected from 5 different people without giving them any prior knowledge on the system. The generated NLU data was then reviewed and added to the already existing one.

During this first test, users shown intents that were not implemented, therefore functions to handle those cases and new NLU data were added. Consequently, a second round of data collection, following the same modalities, was performed on 6 people: 4 of them which also participated in the first test, while the remaining 2 were newly introduced to the system, again without giving any prior knowledge. Once again, the training data was reviewed and used to extended the existing one. Finally, intent examples were augment with the *ContextualWordEmbsAug* and *SynonymAug* functions from the *nlpaug*[6] python library and a set of results were handpicked to expand the data. In addition, lookup tables and synonyms were used to retrieve the correct information value. Additional data was manually added after evaluating the final system. Summarizing up, the data consists of 24 distinct intents, 583 intents examples, 13 entities (9 user-fillable slots), 219 entity examples and 34 stories with an average of 12 turns per dialogue (over 5 stories collected in the second human test) .

To simulate the API information retrieval, mock classes and functions were created. The list of shops and relative menus were manually written with pseudo-real information: 3 famous different restaurants are available and each one comes with a telephone number, a list of available drinks and a list of available pizzas (all with a set of ingredients, price and typology). To avoid increasing the complexity, each pizza or drink object is shared among the restaurants menu (same ingredients and prices), but available pizzas differs between restaurants. A total of 14 pizzas, 17 ingredients and 3 drinks are available. All this domain data is handled by the **PizzaStoreHelper** mock class and relative functions.

4 Conversation Model

The system is powered by Rasa. Hence, the model[5] makes use of a set of pipeline components and a set of policies to process the user input and decide the action to take, respectively. The pipeline uses the following components:

- **SpacyNLP:** the powerful language model used to initialize spaCy structures. As suggested from rasa, the ”medium” size model was used.
- **SpacyTokenizer:** to create tokens using spaCy.

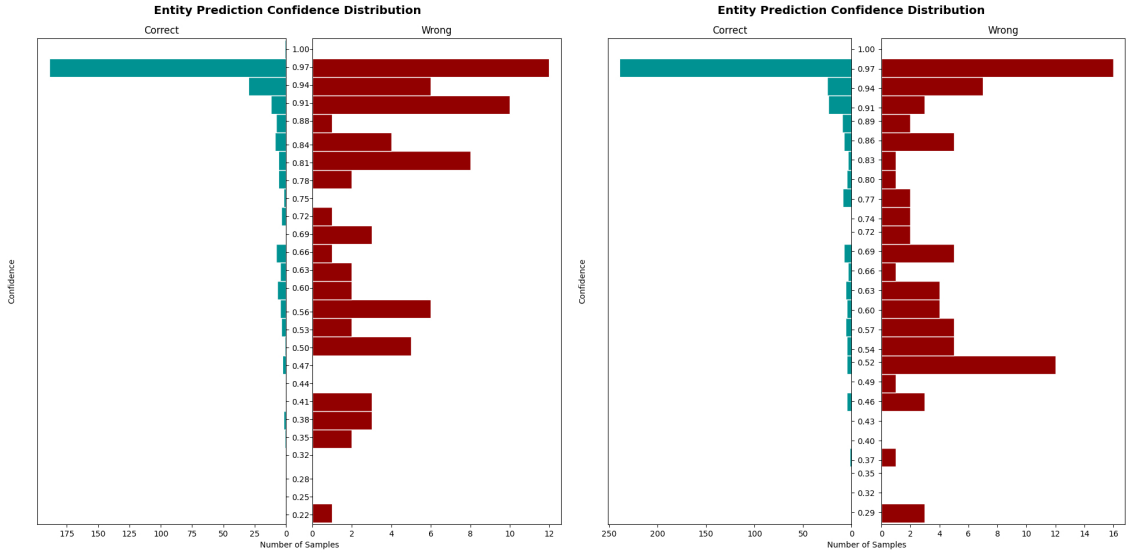

Figure 1: Comparison between the final model trained with cross-validation on two NLU data. On the right, the final intent examples were increased by 10%. More entities are correctly classified but there are still errors, adding more examples should improve the results.

- **SpacyFeaturizer:** to create a dense vector representation of the user messages.
- **RegexFeaturizer:** necessary to use lookup tables.
- **LexicalSyntacticFeaturizer:** allows to create lexical and syntactic features to support entity extraction.
- **DIETClassifier:** the Dual Intent Entity Transformer, used for intent classification and entity extraction.
- **EntitySynonymMapper:** to retrieve the correct entity value if a synonym is given.
- **FallbackClassifier:** to let the user know that the intent was not clear. In that case, a default fallback action is triggered.

The policies consist of:

- **MemoizationPolicy:** allows the system to ”remember” the training stories so to predict the same action if the current conversation matches another in the data.
- **RulePolicy:** to enable ”rules” which allows to specify a standard action given a certain intent.
- **TEDPolicy:** the Transformer Embedding Dialogue policy that helps in predicting the next action according to the current messages history.

Furthermore, the system was deployed on the Google Assistant, therefore it makes uses of their vocal model for speech-to-text and text-to-speech conversion. The procedure is available in the instructor's notebook[3].

5 Evaluation

To improve and get a picture of the performances of the system, different tests and evaluations have been conducted, in particular about the NLU data, model configuration and human usability.

Following the **conversation-driven development** practice[1], 2 rounds of human test were conducted during the development of the system. Given the fact that in the second test the bot was able to satisfy the users, stories collected in this last experiment were used to train the model (see chapter 3 for more details).

Consequently, the model configuration parameters for both the pipeline and policy components were hypertuned using the **rasa test**[8] command. Figure 2 shows the result of this operation.

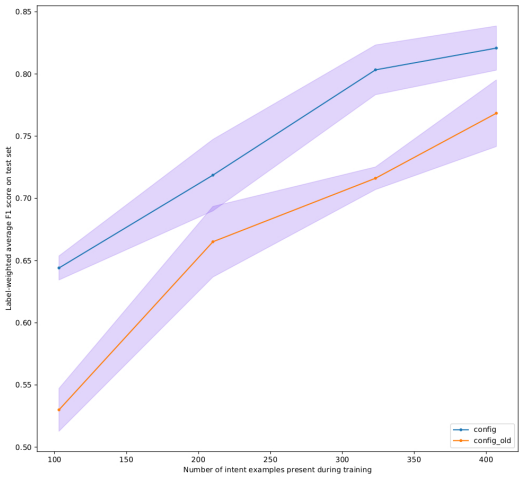

Figure 2: F1 score on test set for a starting model (bottom line, orange) and the final model (top line, blue).

Additionally, a lack of training data was noticed during the process, hence new intent examples were added in order to cover a larger set of possible entities values (see Figure 1 for details).

Finally, the model was trained and deployed on Google Assistant for a final real usage test; 6 people engaged a conversation with the system: 4 of them already used it on both the reviews, 1 only on the first test and the last person using it for the first time. A set of questions were asked to each participant after interacting with the system, here is the list with the average result:

Question	Result
Were you able to order?	100%
Did the system satisfy your requests?	83%
Would you use this system?	100%
Would you recommend the system?	83%
How difficult was ordering? (1 - 10)	1.6
How robotic felt the process? (1 - 10)	7

6 Conclusion

In the previous sections we have seen how a Rasa assistant for ordering pizzas has been designed, implemented and tested. Overall, the final results were quite good and people found it both useful and interesting. Despite that, many improvements can be done including collecting more conversations in order to enlarge the data, let the model generalize better and discover new relevant intents or implementing new functionalities such as ordering a custom pizza or adding/removing toppings to let the user have a more realistic experience.

References

[1] Conversation-driven development. https://rasa.com/docs/rasa/2.x/conversation-driven-development. last access 29/28/2022.

[2] Fallback. https://rasa.com/docs/rasa/2.x/fallback-handoff. last access 26/28/2022.

[3] Integrating rasa with google assistant. https://github.com/esrel/LUS/blob/master/notebooks/rasa_google.ipynb. last access 30/28/2022.

[4] Interactive learning. https://rasa.com/docs/rasa/2.x/writing-stories/command-line-interactive-learning. last access 29/28/2022.

[5] Model configuration. https://rasa.com/docs/rasa/2.x/model-configuration. last access 29/28/2022.

[6] nlpaug. https://github.com/makcedward/nlpaug. last access 31/28/2022.

[7] Rasa. https://rasa.com/. last access 26/08/2022.

[8] Testing the assistant. https://rasa.com/docs/rasa/2.x/command-line-interfacerasa-test. last access 29/28/2022.