NLU project - The restaurant BOT

Paganin Martina
Università degli studi di Trento
martina.paganin@studenti.unitn.it

Abstract

The aim of this Language Understanding project is to build a bot, that is a conversational software capable of understanding user requirements in order to return some suggested restaurants.

The module is built with the help of *Rasa NLU (version 0.12.3)* and *Rasa Core (version 0.9.3)*. Rasa NLU is an open-source tool useful for the classification of intents and the extraction of entities from unstructured text. The Rasa Core framework can be think as the ”brain” of the bot, since it takes as input the information extracted by Rasa NLU to build a dialogue model.

The project is based on the knowledge provided during the Language Understanding Systems course hold by professors G. Riccardi and E. Stepanov (spring 2018)[9].

1 Introduction

This paper describes how the restaurant search bot works and how it was built. Moreover, it will be explained what are the implementation choices taken during the building of such a bot and the motivations behind such choices. The bot is based on the restaurant example bot, but some improvements were added, such as the wider range of possible sentences and the database of restaurants on which the bot can perform its research.

In this project, two steps in the Language Understanding System Cycle are treated: the **natural language understanding** step and the **dialog management**.

Initially, the NLU module (developed by Rasa NLU[8]) will be described, then, this section will be followed by the description of the dialogue model, which is instead mainly developed within the Rasa Core[7] framework.

1.1 The NLU phase

The Rasa NLU library is useful to interpret text: user's sentences are mapped into a structured text which encodes the **intent** of the sentence and extracts the **entities** (concepts). The intent of a sentence denotes its generic purpose. This first phase is useful to the Rasa Core module, as it takes as input intents and entities coming from the Rasa NLU parsing phase. There are many different ways for users to say the same thing, so, it is necessary to abstract and map them into a more generic word. For example, sentences for greeting are labeled with the intent *greet*, while questions about restaurants are tagged with the *inform* label, and so on. The first step Rasa NLU has to do is a classification task: every sentence is tagged with a label coming from a finite set of intent labels, specified in the domain file.

The domain is described in a file structured in the following way:

- slots : the list of entities that has to be stored during the conversation
- entities: the concepts that the NLU model has to extract
- intents: what the user is expected to say
- templates: examples of sentences that the bot can say
- actions: what the bot can do and say

Another important step done by Rasa NLU is extracting the most important words from the sentences: the entities. For example, in this case we are dealing with sentences which express questions about restaurants, hence it will be necessary to extract words about the type of cuisine, the location of the restaurant and the price range.

In order to train the NLU module, a *training set* is provided. Every sample contained in such a set is described by 3 features:

- text: which contains the sentence that has to be interpreted
- intent: the main label, which explains the aim of the sentence
- entities: a list of important words included in the sentences

In order to accomplish such phase, a **processing pipeline**[6] needs to be specified. This contains a list of components for sentences processing through which every sample needs to be parsed. These components operate tokenisation and featurization phases in order to finally perform entity extraction and intent classification.

In this project the entity extraction phase was performed by the Named Entity Recognition extractor, based on **CRF**[4], which is basically a statistical model which predicts label for words by taking into account the"neighborhood" or, more specifically, the context of the sentence. This linear chain can predict the most probable set of entity tags for each input sentence. An alternative entity extractor was also tried (the **Sklearn extractor**), as explained in the NLU evaluation section (3.1).

1.2 The dialogue model and Rasa Core

The *dialogue model* is the core of the bot. After understanding what the user has text, the bot has to select the correct answer. So, some data are needed to train the model and make it capable of deciding what is the next action to do. This is performed by taking into account the current state of the conversation and the previous states, and given that, the most probable action can be taken.

Some considerations about the current dialogue model follow. The bot is designed to manage a simple conversation with human beings, on few characteristics of restaurants such as the type of cuisine, location and price range. This dialogue model only considers the context of searching a restaurant. No context switching is expected from the user, so there is no way to handle such a situation. In the case, the bot may answer in an unpredictable way.

Generally, the user starts the conversation greeting the bot, which proposes itself to help and keep listening to the user's commands. After a first user's statement, the bot takes the initiative, by keep answering questions so it leads the conversation. The bot needs to ask more information in order to get enough information to help the user finding the correct restaurant.

1.3 The database

The information retrieved by the bot are stored in **MongoDB**. As the restaurant database is a very simple one, it was thought that MongoDB was the most suitable platform to support that database. Moreover, MongoDB functions can easily be integrated with python scripts, so querying the database is very fast and natural. MongoDB is a not-relational DBMS: documents are stored in JSON format, without the concept of relations between tables. The restaurant database needs to store only few information, such as the name of the restaurant, the location, the type of cuisine and the price range, hence, a collection with these fields was created.

Queries on the database are made inside the function **RestaurantAPI**, called by the action **SearchRestaurant**. During the conversation with the bot, some information have to be kept in memory, so data are automatically stored into special locations called **slots**. SearchRestaurant extracts these data and makes a query on the Restaurantdb, through the RestaurantAPI. Results of the query are then stored in the "matches" slot and passed to the **ActionSuggest**, which has to show the result to the user.

Queries on the database search for a specific cuisine, city or price range. If any of these parameters are missing, the remaining are still used to perform the search. If the query is not able to find any matching restaurant, the price range parameter is removed from the query, and another attempt is tried. If the query result is still empty, also the type of cuisine is ignored, and finally if there are no suggestions, one random is returned.

When the bot runs, *actions* are chosen by considering the actual state of the conversation. Policies are very important in this step, because they compute the probability of actions and they select the one with the highest value.

There are different kinds of *policies*[5] (section 1.4), and they allow to obtain different results, which will be instead discussed in section 3.2.

1.4 Policies

The most simple approach uses a Sklearn classifier to train a policy. The classifier based on logistic regression uses a grid search with some different C parameters to choose among with. In order to choose the best value, a cross validation test phase can be performed.

The Fallback policy[3] is deterministic. This policy executes a specified action if the confidence of an intent prediction is lower than a given NLU threshold or if any of the other policies was able to predict an action with higher threshold. In such a case, the bot will be set to do the default action with probability equal to 1. Otherwise, if the confidence is higher than the NLU threshold, the probability of taking the fallback action (listening) will be set to a specified value.

The memoization policy just memorizes the actions taken in the stories. Examples of conversations are given to the policy and without using machine learning it simple memorizes the stories. A variant of this policy is the AugmentedMemoization which also ”remembers” slots together with actions, allowing to preserves information stored into slots during the predictions.

The Keras policy is instead based on the Keras neural network library, so it uses machine learning to train the dialogue model. The Restaurant custom policy is an alternative of the Keras one, since it extend that class by implementing a different model architecture.

1.5 Actions

Bot can do different things: the list of actions[2] which it can perform is specified in the domain file. With actions, bot can answer to users, query a database and call external APIs. There are different kinds of actions. The most simple one has just to answer message to the user, but there can be also more complicated actions. This bot has to do two actions: search restaurants in the database (implemented in ActionSearchRestaurants) and show the results to the users (defined in ActionSuggest). Both these actions are defined in bot.py python script.

With Rasa Core 0.9.3 a special kind of action class can be used: the FormAction[1]. In a form action a set of desired fields can be defined, hence a single action can be used to obtain answers for more than one question. If during the conversation the bot detect that one piece of information is missing to complete the database query, it asks the user for it. The set of required fields is composed by the requirements necessary to perform the search of a restaurant:

- cuisine
- location
- price

2 Data Analysis

To train and test the NLU component of this project, the franken dataset was used. This dataset contains sentences pronounced when information about restaurants are required. So, the most frequent kind of conversation is based on a dialogue of this type: greet the bot, ask for information about restaurants (location, type of cuisine and price range), affirm and/or deny to the answers provided by the bot, and finally, thank it and leave the conversation.

The franken dataset consists of 1977 sentences, each of them is labeled with a specific word which denotes the intent of such sentence. There are six types of intents:

- affirm
- deny
- greet
- inform
- request information
- thank you

As can be seen in the following table (table n.1), the 50% of sentences is of type inform. This kind of statement needs to be the most frequent one, as there are many different ways of asking questions about restaurants. These sentences are used to ask for details about restaurants, such as the type of cuisine, the telephone number, the address and the price range. These information (asked by the user) are memorized during the conversation in order to make a specific database query and retrieve the restaurant which is suggested to the user.

Moreover, entities are extracted only from sentences whose intent is inform and request information, due to the fact that affirm/deny/greet sentences does not contain any entity useful for the search of restaurants. Entities extracted contain

Intent	Frequency
Affirm	260
Deny	85
Greet	13
Inform	1014
Request info	16
Thank you	589
Total	1977

Table 1: Intent distribution

Entity	Frequency	Total # of instances
Cuisine	70	573
Info	2	16
Location	9	338
People	4	12
Price info	6	354
Total		1293

Table 2: Intent distribution

information about details of restaurant, and they are listed in table n.2.

To train the dialogue model, a different dataset was necessary, so the babi stories dataset was used. This dataset contains 1000 stories, which are particular data samples, representing examples of training conversations. A story describes how dialogues between users and bot work. Every story is composed of a sequence of user intent, followed by actions taken by the bot. An example of story can be the following:

- user starts the conversation greeting
- bot answer offering to help
- user ask for a restaurant
- bot ask details for the search
- user answers
- bot searches and proposes some results
- user can confirm or deny
- bot can keep on searching, asking other requirements
- user thanks
- bot says bye

Stories are then used to train the policy, in order to teach the bot how to respond to user requests.

	NER CRF	NER SPACY
Accuracy	0.99	0.985
F1 score	0.99	0.984
Precision	0.99	0.985

Table 3: Evaluation NLU model

3 Evaluation

3.1 NLU module

Two variants of pipeline for the NLU phase have been tried, both based on Spacy. A pipeline is basically a list of components used to process sentences, in order to map them into intents and extract entities.

The first pipeline tried, uses a spacy tokenizer, followed by an intent featurizer and a classifier. Then, entity extraction is performed by using CRF. Finally, entities with different values but same meaning are mapped into synonymous values.

In the second pipeline, the CRF entity extractor was substituted with the Spacy entity extractor, which is based on an averaged perceptron. From the evaluation results showed in the table above (n.3), it can be seen that CRF performed highly better than the Spacy extractor.

The NLU component was tested with 10-fold cross-validation.

3.2 Dialog Manager

After some tests, it resulted that the best policy combination was the AugmentedMemoization + Sklearn. No combinations with other policies allow to obtain sensible improvements.

Several attempts were made to find which was the best maximum number of stories to train the Memoization policy, but this technique alone does not give acceptable results. Also the custom RestaurantPolicy, if used alone, does not give acceptable results. If these two techniques are combined together, better results can be obtained, but still not optimal.

The fallback policy alone performs very poorly, and when combined with other techniques no sensible advantages were noticed.

Results obtained from this test phase are listed in table n. 4, which shows the averaged F1-score value for the most relevant policies used.

At the end, it comes that some sample actions are not correctly detected: the confusion area is localized in the *action_listen* row: this happens when

Policy	**Average F1**
Augm.-Sklearn	0.924
Augm.(10 max history)-Rest.	0.501
Restaurant	0.537
Sklearn	0.840

Table 4: Evaluation dialog model

the action of listening is wrongly classified with the *None action*.

4 Results and discussion

In conclusion, it can be said that Rasa NLU performs in a very good manner, due to the high number of training samples and the small size of the intent set. It comes that intents of sentences and entities are correctly detected and extracted.

Moreover, Rasa Core policies performs very well, in particular the AugmentedMemoization policy. Better results can be obtained by combining policies, because each of them behaves singularly in a particular way, so advantages of techniques can be merged together to increase performances.

Finally, some considerations on possible future improvements. First of all, speaking to the bot and making it speaks should be the first feature that has to be added.

A more natural conversation should also be developed, so sentences which make the user understand that the bot has received the correct information have to be added, in order to help the user rectify the conversation.

In addition, extending the bot actions by managing some more information should be nice, such as asking more details about the restaurant, maybe how to reach it or other users' opinions about it.

References

[1] Action form. https://core.rasa.com/patterns.html. Accessed: 2018-06-0.

[2] Actions. https://core.rasa.com/domains.html. Accessed: 2018-06-0.

[3] Fallback action. https://core.rasa.com/patterns.html#fallback-default-actions. Accessed: 2018-06-0.

[4] CRF entity extractor. https://nlu.rasa.com/0.12.3/entities.html. Accessed: 2018-06-0.

[5] Policies. https://core.rasa.com/policies.html. Accessed: 2018-06-0.

[6] Processing pipeline. https://nlu.rasa.com/0.12.3/pipeline.html#section-pipeline. Accessed: 2018-06-01.

[7] Rasa Core. https://core.rasa.com/index.html. Accessed: 2018-06-0.

[8] Rasa NLU. https://nlu.rasa.com/0.12.3/index.html. Accessed: 2018-06-0.

[9] E. Stepanov G. Riccardi. LUS course, spring 2018, unitn.