Language Understanding Systems — Final project - Dialog System within Rasa framework in movie domain

Anonymous ACL submission

1 Introduction

The objective of the final project is to create a dialogue system with *Rasa* framework in movie domain. So, the bot needs to answer the users' questions and to keep track of the discussion in order to give additional information. The bot is developed in python and the database is MySQL.

2 Rasa data and files

Rasa is a framwork that allows to create and train a NLU and to create a dialogue manager that handle a conversation. To do so it is necessary to create three files that contain:

- training data for the NLU;
- information about the domain;
- stories for dialogue training.

2.1 From NL-SPARQL dataset to NLU Data Format

The training file for the NLU is a JSON containing sentences, with the related intents and the entities, if there are any. Since the starting dataset is the same of the previous project, it needed some modifications. So, it was created a script in python that automatically makes the conversion from a format to the other. In this section are described the operations for the conversion.

2.1.1 Entities

An **entity**, in Rasa, is the meaning that a list of one or more words have for the bot in a specific domain. In this case an entity could be the name of a movie or a specific year. The original dataset is formed by many sentences and each sentence is divided in as many lines as the words. Each word has its corresponding IOB-tag. The tags can start with an I, an O or a B, respectively "Inside of span","Outside the span" and "Beginning of span". For this task the words with the tag "O" are not usefull, because they don't carry any important information for the NLU. However, the tags "B" and "I" mean that the related word is part of an entity or it is an entity itself. To extract an entity, the script reads every line until the "\n" character, which signals the end of a sentence, and for every tag that encounters it creates the related entity. When the sentence is over and all the entities are extracted, it searches their position inside the sentence.

2.1.2 Intents

At the end of a sentence the script looks also for the intent in the labels file, since the labels are the intent themselves, and it put it with the example inside a dictionary. An **intent** describes what the users probably want to know. Besides the labels, there are other intents listed in the domain file. These are represented by some examples written by hand or found in some tutorials by Rasa.

2.2 Domain information

The domain is a YAML file that contains:

- slots
- entities
- intents
- templates
- actions

The entities and the intents are the same as in the training file. The information the bot needs to keep track is stored in the slots, which have a specific type such as "text" or "list" or "float". The actions are what the bot can actually do. So, an action can be just a simple answer to a greeting, in which case

the action name starts with "utter_" and the name of the action, or can be a more complex one. Even though the actions are listed in here, they are implemented in a *python* file. At last, the templates are exactly the messages that the bot uses to answer the basic messages of the user.

2.3 Stories

The stories are the examples that the dialogue manager needs to learn how a conversation can take place. A story has in order: a title, the user's intent followed by zero or more entities and the bot's actions followed by the slot, if any were filled. To build a solid dialogue system can be useful to add some other intents after the bot actions, creating an example of full conversation. Starting from an initial set of stories written by hand, some others were added during testing phase with the online training method, which will be described in the paragraph "Dialogue training" in the third section.

3 Bot development

The assistant bot developed for this project has three main files, *database_manager.py*, *bot.py* and *actions.py*.

3.1 Database Manager

In order to get some answer for the user the bot needs to query the movie database and to do so it creates an object called MovieBuff, which is the class that has all the methods to query the database. There is a select query builder that takes in input an indefinite number of parameters and it build the query adding more conditions based on the parameters that have a value. Since the user can type the name of a movie in a wrong or incomplete way, or even if it is written without capital letters, a select that uses the equal operator is always returning zero rows. The "LIKE" operator solves this problem, but it returns a higher number of rows in which the correct result isn't always in the first line. Thus, before returning the result, the MovieBuff class calls a method that look for the closest match and it returns it.

3.2 Actions

The actions listed in the domain file are implemented in this file. They are all subclass of the Rasa class *Action* and they have two methods that have to be implemented, **name** and **run**. The first method just needs to return the name of the action that will be used in the stories. The method "run" is the one called when the bot makes the action.

Initially there were an action for every intent found in the dataset, but some of them can not be implemented, because in the database there are not all the information that a user may want to know. Thus, some of the actions were thrown away and the JSON file for the NLU was modified changing the related intent in "other". The dataset provided also some labels with multiple intents. Since Rasa does not support multiple intents, the adopted solution is to change the white spaces in "_and_" in order to form a single intent.

The evaluation of the NLU with the new intents (Tab. 2) shows that the majority of them (the bold ones) are not well recognized from the NLU and so they were changed in "other". In conclusion, the remaining actions are *movie_and_director*, which shows a list of movies for each director, and *movie_and_revenue*, which shows a list of revenues for random movies or for a specific director.

3.3 Bot

The training and the usage of the bot is divided in three different functions, *train_nlu*, *train_dialogue* and *run*. While the last one is just the loading of the trained model, the other two functions are less trivial.

3.3.1 NLU training

when creating a new Trainer object the configuration file with the pipeline is passed as argument. Then the NLU is trained with the training data created before and at the end the model is saved inside a folder. At this point the NLU can tell what is the user intent when it receives a message.

3.3.2 Dialogue training

After the training, the NLU could be ready to receive questions and give answers, but in order to have a conversation it is necessary to train an agent that handle the dialogue. The agent takes in input the interpreter (the NLU created), the domain file and two policies, Memoization and Keras. Then the agent is trained over the stories and it is saved inside a folder. For testing purposes there is also a method called "online_training" that for every message received shows what the bot understood, what is the user intent, which entities are recog-

nized and what is the next action that the bot think it should make.

4 Evaluation and results

The NLU was evaluated with the Rasa NLU library and the results are showed in Table 1 and they are all over 90%. The best method to evaluate the bot is to use it and see how it behaves. Even if the accuracy is high, the bot has some limits. One is the wrong recognition of a movie name. Sometimes the name of a movie is only recognized by half, for instance, “men in black“ is splitted in two: “man“ as the movie name and “black“ as the language. The dialogue is handled well, because if the user search for a director name and then the release date the bot provides the correct answer, but there is a case where in a sentence “it“ is used as pronoun, but it is recognized as a movie title.

Table 1: NLU evaluation results

F1-Score	0.928265461144
Precision	0.921879261649
Accuracy	0.941881366087

Table 2: Evaluation of the NLU with spacy pipeline

	precision	recall	f1-score	support
actor_name	0.95	1.00	0.98	202
affirm	0.92	1.00	0.96	11
award_ceremony_and_award_category	**0.00**	**0.00**	**0.00**	**2**
budget	0.99	0.99	0.99	113
country	0.98	1.00	0.99	60
deny	1.00	1.00	1.00	3
director_and_movie_and_date_and_star_rating	**0.00**	**0.00**	**0.00**	**1**
director_and_movie_and_star_rating	**0.00**	**0.00**	**0.00**	**4**
director_and_movie_name	**0.00**	**0.00**	**0.00**	**1**
director_name	0.95	0.98	0.97	178
genre	0.97	1.00	0.98	61
goodbye	1.00	1.00	1.00	8
greet	1.00	1.00	1.00	8
language	0.92	1.00	0.96	49
movie	0.93	0.93	0.93	1527
movie_and_budget	**0.00**	**0.00**	**0.00**	**1**
movie_and_date	**0.00**	**0.00**	**0.00**	**2**
movie_and_director	1.00	0.83	0.91	6
movie_and_language	**0.00**	**0.00**	**0.00**	**1**
movie_and_rating	**0.00**	**0.00**	**0.00**	**3**
movie_and_revenue	1.00	0.33	0.50	3
movie_and_star_rating	**0.57**	**0.25**	**0.35**	**16**
movie_count	1.00	1.00	1.00	30
movie_name	0.77	1.00	0.87	24
other	0.85	0.83	0.84	675
rating	0.94	0.99	0.96	75
rating_and_star_rating	**0.00**	**0.00**	**0.00**	**1**
release_date	0.93	0.99	0.96	163
revenue	0.98	1.00	0.99	130
runtime	1.00	1.00	1.00	7
star_rating	0.00	0.00	0.00	3