Building chatbots for customer support: fast and serious

Diogo Fernandes
diogobfernandes@tecnico.ulisboa.pt

Instituto Superior Técnico, Lisboa, Portugal

October 2021

Abstract

Conversational Agents are systems that are used in a wide variety of areas to assist the accomplishment of a task by the user. However, creating a conversational agent usually requires a great amount of data, which difficulties the development process to create agents for new tasks. To tackle this issue, we investigate how to expedite the development process. Firstly, we provide two separate frameworks that are able to create conversational agents in Rasa using the information available in the MultiWOZ and Taskmaster-1 datasets. We conduct an automatic and a user evaluation, which show that it is possible to expedite the creation of conversational agents using datasets, highlighting the importance of annotating information such as the slots, the correspondent values, the user utterance intents and also the information required by the user. Lastly, we provide a framework that using a Rasa conversational agent created for the restaurant domain, is able to create a conversational agent for the hotel domain. Our results suggest that it is possible to expedite the development of conversational agents by creating agents for new tasks using information from a preexisting conversational agent.
Keywords: Conversational Agents, Rasa Open Source, MultiWOZ, Taskmaster-1, Knowledge transfer, Development

1. Introduction

Conversational agents, also known as dialogue systems, have been a hot topic in Natural Language Processing in recent times and can be divided in two types: chit-chat (conversation without a reason) and task-oriented (the user has a goal in mind). This work focuses on the latter.

Dialogue systems are usually composed by three different modules: a Natural Language Understanding (NLU) module, a Dialogue Manager and the Natural Language Generation component. To create the best dialogue systems, many have tried to improve the quality of each of these modules. For example, Goo et al. (2018) attempted to improve the intent classification and slot filling tasks for the NLU module. For the dialogue manager, rule-based (Habib, Zhang, and Balog 2020) and machine learning based (Li et al. 2017) solutions were proposed. Moreover, many frameworks have been proposed in order to develop conversational agents. Some of these frameworks are Microsoft Bot Framework[1], Amazon Lex[2], Dialogflow[3] and Rasa[4].

However, the authoring of conversational agents faces two great obstacles: first, there is the need of a great amount of training data in order to train the agents on a task. Often, the lack of training data limits the number of tasks the agent is able to fulfill. Second, the behaviour of the agent during the conversation also needs to be defined. This means that a dialogue manager must be able to handle every conversation path the user can pursue to perform a task while engaging with the agent.

As a result of the reasons presented above, developing a conversational agent is often difficult. For these reasons, it is important to expedite in any way the process of developing a conversational agent, which is the focus of this work. To do so, we concentrate on automatizing the development of the NLU module and the Dialogue Manager module. To test this process, we will use the Rasa framework to create conversational agents, which is an open source framework.

To tackle the lack of data, many datasets have been proposed. This is the example of the MultiWOZ (Budzianowski et al. 2018) and Taskmaster-1 (Byrne et al. 2019) datasets, that contain task-oriented dialogues for several domains, developed using Wizard-of-Oz approach (Budzianowski et al. 2018). In both datasets, the dialogues are annotated. However, the annotations in each of the datasets is different, which leads to a different process of using the available information to create a

[1] https://dev.botframework.com/
[2] https://aws.amazon.com/pt/lex/
[3] https://cloud.google.com/dialogflow/docs
[4] https://rasa.com/docs/rasa/, version 2.2.0

conversational agent. With these datasets, we want to verify what is the process required to create a conversational agent, and if the information in each of the datasets is enough to create a complete dialogue system.

Furthermore, another possibility to expedite the development of conversational agents is reusing existing conversational agents. We believe that it is possible to transfer the knowledge gathered from one domain to create a conversational agent in a different domain. In this work, we conduct an experiment on transferring the knowledge acquired from the restaurant domain to the hotel domain.

2. From the MultiWOZ dataset to Rasa

In this section, we address what is the information required from the datasets to develop conversational agents using the MultiWOZ dataset. In the process, we show the details of MultiWOZ, and explain our process to transform the dataset in a Rasa agent.

2.1. The MultiWOZ Corpus

MultiWOZ is a dataset that contains dialogues created using the Wizard-Of-Oz approach and it has conversations from 8 different domains, where one dialogue might have messages regarding a single domain or more.

Consider Table 1 for illustration purposes. Each dialogue is composed by a set of turns, the user turn and the assistant turn. In both the user and assistant turns, the identity of the speaker and the correspondent utterance is given. Furthermore, an updated state of the dialogue is also given in every user turn. This can be seen in Table 1, that in the second user utterance, the state contains both the information given in the previous turns and the restaurant-food slot it just mentioned.

The dialogue state is composed by three different types of information. First, there is the Active Intent. The Active Intent represents the general task and objective that the assistant is trying to achieve. Moreover, we also have the slots, which represents all he information the user has given and can collect in the the course of the conversation. For last, MultiWOZ have requested slots, which represent information the user asks for in a turn.

2.2. Creating Domain and Adding Training Data to Rasa Agent

To create a Rasa agent, the definition of the domain, as well as adding training data are mandatory. Since the information for both is analogous, the retrieval of information results from a single analysis of the dataset dialogues. For each user turn, the state and utterance of the dialogue is analysed and gives information about certain aspects.

2.2.1 Creating Rasa Intents

MultiWOZ gives in each state what is the the Active Intent. Also, the Active Intent can change during the conversation. For example, the user can start by looking for a place to stay (**find_hotel**), and then proceed to make a booking for place (**book_hotel**). During this section, we call the Active Intent as the task of the user.

Every time the task in the dialogue changes, there are two steps we perform: create the task as a Rasa intent if it was never seen before, and label the utterance as part of the same intent. Some intents and labeled utterances are **book_train** with the utterance *"Yeah, can you book 4 tickets for me?"* and the intent **find_taxi** with the utterance *"I'll also need to get a taxi to go between the 2 places"*.

Take Table 1 as example. In many of the user turns, the utterance contains new information that is important for the task. For that reason, we create a new intent: the **inform** intent that is responsible to give more information to the agent. Any utterance where the task is the same of the previous state, then it is labeled as part of **inform**. In Table 1, since the task is the same in the first and second user turns, then we label the second utterance as **inform**.

However, not all utterances where the task did not change had additional information. Take Table 2 as a reference for the rest of this subsection. Sometimes the user could ask for a recommendation (1), affirm (2), deny (3), request information (4), or at most cases, give information (5). Since each of these lead to different actions by the assistant, then there is the need to differentiate all these utterances.

To differ utterances where the user gave additional information, we take information from the updated state of the dialogue. If there are new slots in the current utterance, such as utterance 5 in Table 2, then we know that the utterance contains new information, labeling it as **inform**.

Moreover, the same logic applies to utterances where the user requested information, since the dataset contains the requested slots and these are updated in the state, such as utterance 4 in Table 2. For this reason, we create the **request** intent. If the user asks for information instead of giving to the user, the utterance is annotated as **request**.

For the utterances like 1, 2 and 3 in Table 2 we decided to not add them to the training data, since it would add noise to the agent, and there was no information in the dataset to indicate the intention of each of them.

For last, at the end of the dialogue, there are utterances where the task is **NONE**. For this reason, we assume that every user utterance at the end of the dialogue is for the purpose of ending the dia-

Utterance	Dialogue State	
USER: Can you help me find an expensive restaurant in the west?	**Active Intent**	find-restaurant
	Slots	restaurant-pricerange: expensive restaurant-location: west
ASSISTANT: Yes, what type of food are you looking to eat?	-	
USER: I would really like to have Indian tonight.	**Active Intent**	find-restaurant
	Slots	restaurant-pricerange: expensive restaurant-location: west restaurant-food: indian
ASSISTANT: tandoori palace matches your criteria. Would you like to make a reservation?	-	
USER: No, but could you give me the phone number, please?	**Active Intent**	find-restaurant
	Requested Slots	restaurant-phone

Table 1: Example of a dialogue state being updated after the user gives information over multiple turns and asks for phone number.

#	Utterance
1	Not really. I'm down for anything what would you recommend?
2	Yes that will be fine
3	No, that doesn't matter
4	What is the address, please
5	I would like to leave on sunday and arrive by 17:15

Table 2: Example of followup utterances after stating the intent of finding a restaurant.

logue, thus creating and labeling the utterances as the intent **goodbye**. Some examples of the labelled utterances are "Thanks, that's all I need. Have a nice day" and "Nope. I'm all set. Thanks again".

2.2.2 Slot Recognition and Rasa Entity Mapping

Just as we seen previously, utterances can contain important information to complete the task correctly, which is represented as slots in MultiWOZ. To use this information in Rasa, we create in the domain an entity and slot for each of the slots that exists in the MultiWOZ dialogue. Moreover, during the dialogue, we use the value of the slot and the utterance to annotate it for the training data.

In order to annotate the values of the slots in the user utterance to add them as training data for Rasa, we follow the following steps:

1. Filter all the slots to have only the new ones
2. For each slot, search for the slot value in the utterance
3. If the word in the utterance is the same as the slot value, then replace the word with the form *[slot_value](slot_name)*
4. If the word in the utterance has bigger length than the slot value, the replace the word with *[word](slot_name)* and add it as a synonim to the slot value
5. If the word is not found, then discard the slot
6. If the annotated value is a hour or a number, create a hour regex or a number regex, respectively, for the associated entity.

Using this process, we guarantee that the slot values found in the utterances correspond to the latest turn slots. However, this also leads to incorrect utterance annotations, in cases where the user mentions a slot that has been mentioned in a previous utterance. Moreover, we add the synonyms to add simplicity of dealing with the values later during the execution of custom actions. Furthermore, although the creation of regex limits the speech for the user (for example, the user can say "1" or "one", but regex forces the user to say "1"), this allows that with limited training data, the agent is able to recognize any number, and hour in the form of *hh:dd*.

2.2.3 Mapping Requested Slots to Rasa Entities

Reminding the knowledge of the MultiWOZ dataset, requested slots are slots that appear in the state of a user turn of a dialogue, and represent the information that the user requests in the utterance during the current turn. Moreover, requested slots do not have a value in the user utterance. An example of this slot is *restaurant-phone*. To annotate

these slots, we create a Rasa entity with the name **requested_info**.

In order to annotate the slots in the utterances, we consider the right side of the slot has the initial value. Then, we perform the following steps:

1. Search for the value in the utterance
2. If the value is found, then replace the value in the utterance in the form *[value](requested_info)*.
3. If the word is the utterance has a bigger length than the value found, replace the word in the form *[word](requested_info)*.
4. If it was not found, divide the initial value in two parts, and go back to step 1.
5. It the initial value has already been separated in all possible ways, end the search.

This way, we guarantee that all slot names lead to a match in the utterance. This is the example of the slot **restaurant-phone** where the value "phone" is found in the utterance, or the slot **attraction-entrancefee** where the value "entrancefee" can be found in the utterance, after being divided to "entrance fee".

2.3. Creating Rasa Stories and Rules

In Rasa, a story is composed by a set of the user intent and entities, followed by the action of the conversational agent. In this subsection, we proceed to annotate the assistant utterances by creating Rasa actions and Responses, and then use these to create Rasa stories.

```
- story: dialogue 182
  steps:
  - intent: find_restaurant
    entities:
    - restaurant-name: cafe uno
  - action: action_find_restaurant
  - intent: book_restaurant
    entities:
    - restaurant-bookday: sunday
    - restaurant-bookpeople: 7
  - action: utter_ask_restaurant_booktime
  - intent: inform
    entities:
    - restaurant-booktime: 16:15
  - action: action_book_restaurant
  - intent: goodbye
  - action: utter_say_goodbye
```

Listing 1: Example of a designed story using our solution

Take Listing 1 as an example. The story represented in this listing starts by the user stating its will to find a restaurant called *cafe uno*. The agent gives the information about the restaurant and the user proceeds to make a reservation, giving the information about the *day of booking* and also the *number of people*. As there is information missing, the agent asks for the time of booking. The user gives then the missing information and then the booking is completed by the assistant. The dialogue ends just as the user states its intention to terminate the conversation.

To create this story, we make two assumptions: first, we consider that for every Active Intent in the dataset, there is an action with the name *action_active_intent*, for example **action_find_restaurant** and **action_book_restaurant**. Each of these actions is responsible to fulfill the requests of the user and also make any verifications that are required.

However, the assistant will not always have all the information required to fulfill a task. For example, to make a reservation for a restaurant, the assistant needs to know the day and time of the reservation, as well as the number of people. For this reason, we create a Rasa response if any of the requested slots for completing the task is missing. This process is repeated for each of the valid dialogues of MultiWOZ, creating the number of conversation paths equal to the number of valid dialogues on the dataset. We consider that a dialogue is valid if all the user utterances have been labeled correctly with Rasa intents (utterance labeling is explained in Subsection 2.2).

However, the dataset does not contain what are the requested slots. For this reason, we divide the dialogues in tasks, and calculate what are the mandatory slots for each task using the following formula:

$$a\%(s, task) = \frac{\#a(s, task)}{\#dialogues(task)} * 100$$

For each slot, we calculate the number of times it appears in the dialogues of a task, divide it by the total number of the dialogues of the same task and multiply the result by hundred. This result represents the percentage of task dialogues where the slot appears.

After calculating this value for every slot at every task, we compare this value with a threshold value. If the value of the appearances of the slot is above or equal the threshold value, then the slot is considered a mandatory slot for the task. We adjusted the threshold value to 85%, which gave us the best results. For example, the mandatory slots for the task of **book_train** are: the *train_day*, the *train_departure*, the t*rain_destination*, and the *train_bookpeople*.

Given that we already determined what are the mandatory slots for each dialogue, the selection of the action for both a task-specific intent and the inform intent we defined in the previous section for the story becomes simple. If the Rasa intent of the user is a task-specific intent, or the **inform** intent, then we check if all mandatory slots have been given. If not, then we select the action of the assistant to be of the type **utter_ask_slot_name_other_slot_name**. Otherwise, the action selected is **action_task**.

For last, we create a simple response in the form of **utter_say_goodbye** for the **goodbye** intent. Also, for the **request** intent, instead of adding this intent to the stories, we create a Rasa rule, since the action performed by the agent in the presence of this intent is always the same, regardless of the conversation history. For this intent, we create the action **action_get_requested_info**. The rule created for this intent is in Listing 2.

```
- rule: give requested info
  steps:
  - intent: request
  - action: action_get_requested_information
```

Listing 2: Rule created for the handling of the intent request.

Concluding, we were able to define all the properties required to create a conversational agent in Rasa. In order to prepare the agent for training and execution, we only to define the text for the Responses, as well as the configurations and the implementation of the Custom Actions, which can be found in the repository[5].

3. Knowledge Transfer

In this section, we continue the research on what is the important information from datasets using the Taskmaster-1 dataset to create a conversational agent in Rasa. While creating the conversational agent using Taskmaster-1, we also address the possibility of using the conversational agent created from MultiWOZ to improve the dataset information. Later in this section, we explore how can the knowledge learned for one domain can be used in another domain.

3.1. Adapting Taskmaster-1 Dataset to Rasa

The Taskmaster-1 dataset, just as MultiWOZ, is a dataset that contains task-oriented dialogues. The dialogues were created either using a self-dialogue or a Wizard-Of-Oz(Byrne et al. 2019) approach. Moreover, each dialogue can belong to one of six domains: car repairing, restaurant reservation, ordering movie tickets, ordering coffee drinks, pizza delivery and taxi service. We focus on the restaurant reservation domain. Thus, examples and assumptions done on this point forward are only in regard to the restaurant domain.

Take Table 3 for illustration purposes. In Taskmaster-1 dialogues, in each turn (user and assistant) it is given the utterance, and information about the slots. The information given by a slot is its value, which is a part of the utterance, and the annotation of the slot. If we look at the annotation, we see that it can be broke down in domain, followed by the slot type, then the name of the slot and, optionally, ends with the acceptance (or not) of the parameter by the agent at the moment of the booking. One detail of the dataset is that only slots that are mandatory to complete a task are annotated. For example, the type of food during the search for a restaurant are not considered as a slot since they are optional in the task of booking a restaurant.

Using this information, we can create the **entities** and **slots** in the Rasa domain, as well as annotating the slot values in the format *[slot_value](slot_name)* to add as training data, since the slot values are always part of the utterance.

Aside from the slots, there is no more information in the dialogues, which hinders creation of a conversational agent in Rasa. However, we noticed that the structure of the restaurant domain dialogues could be broken in two parts: finding and booking a restaurant. Moreover, we also noticed that there were also utterances where the user asks for information (Table 4 exemplifies the cases we mention).

Thus, to find this information, we took two approaches: the first one is by making assumptions to divide the dialogue in parts and labeling the utterances, and the second is to use the the MultiWOZ's restaurant domain agent to understand some intents and extra entities from the utterances.

3.1.1 Creating Assumptions to Label Utterances

To retrieve more information from the dialogue, we performed the following assumptions:

- All the user utterances until the slot **location** is found are merged in one and labeled as **find_restaurant**.
- After finding the location, we label each of the following user messages as part of **inform** until the first restaurant name is given by the assistant.

In the results of the intent *find_restaurant*, it is suggested results suggested that the user had the

[5]https://github.com/Fogoid/MultiwozToRasa

Utterances	Value	Annotation
USER: Tell me good Chinese restaurants in new york	new york	restaurant_reservation.location.restaurant.accept
ASSISTANT: Hakkasan and uptown restaurant Philippe Chow are top rated	Hakkasan	restaurant_reservation.name.restaurant
	Hakkasan and uptown	restaurant_reservation.name.restaurant
	Philippe Chow	restaurant_reservation.name.restaurant
USER: which one is near to airport?	near the airport	restaurant_reservation.location.restaurant

Table 3: Taskmaster dialogue example for the restaurant domain and information available by utterance.

#	Utterance
1	I'm looking for a nice sit down restaurant in San Francisco, California.
2	No, that'd be fine. Let's go with the second restaurant, please.
3	Okay, and is there good actions for kids?

Table 4: Examples of utterances where the user has different intentions. The first and the second are for finding and making a booking for a restaurant, respectively, and the last one is to get information about the restaurant.

intention of finding and later booking a restaurant. However, the second assumption, the utterances we labeled as *inform* could add information to the dialogue, such as the utterance "Okay. I'm looking for a Greek restaurant with Greek food", or add no information, such as "Thank you.", "No problem." and "What is that one?".

We can conclude by this experiment that by having only the information from the slots in the dataset is not sufficient to create a conversational Agent in Rasa without having noise in the training data.

3.1.2 Using MultiWOZ to Add Information to Taskmaster-1

In this subsubsection, we use the MultiWOZ restaurant agent and try to add new information to the Taskmaster-1 dataset, such as the annotation of optional slots, as well as trying to differentiate the utterances labeled as **inform** by the second assumption we described earlier.

To get the information from the MultiWOZ dataset, we apply the agent to each utterance labeled as **inform**, and retrieve the information. We can collect the most probable intent (as well as the confidence of the rest of the intents) and the entities predicted by the agent. Some of the results we obtained are: for the utterances "I was thinking Thai food." and "Okay, is it is it cheap?" , the predicted intent was inform, and no entities were found. Moreover, in the utterance "Okay, what's the price like on that?", the predicted intent was goodbye, and the entity found was requested_info with the value price.

Looking at these, in the first utterance, we expected to extract the **Thai** entity as a **restaurant_food**. On the second example, the utterance but as an inform, which is incorrect. For the last example, the agent is extracts the correct entity, but classifies it as a goodbye, when it is a **request**.

Analysing the results of this experiment, we conclude that the MultiWOZ restaurant agent is insufficient to improve the information in Taskmaster-1 dataset. This is for two reasons: first, the utterances used for the training in the MultiWOZ restaurant agent are not alike the utterances in the Taskmaster-1 dataset. Moreover, it is sure that utterances like *"no problem"* can not be classified correctly which is expected, since this messages are not considered for the training of the agent. For last, we can conclude that the knowledge acquired from the MultiWOZ restaurant agent is insufficient to add information to Taskmaster-1. However, we argue that if the training data of the agent was similar to the Taskmaster-1, then it is possible to add new information.

3.2. Creating New Agent Using Knowledge From the Restaurant Domain

In this subsection, we create a Rasa agent for the hotel domain using the agent previously created for the restaurant domain. With this experiment, we expect to understand if it is possible to improve the development process, by using a conversational agent that was previously created.

For the target domain, we believe that the tasks of the target domain must be compatible with the original domain. For that reason, we chose the hotel domain as the target. First, the number of tasks is the same. Moreover, each task has the same number of mandatory slots.

To make the transfer from one domain to the other, we maintain the **intents**, **responses** and **actions** in the domain, and the **stories** and **rules** we defined. For the **entities** and **slots**, we create a mapping from the restaurant slots to the hotel slots. The mappings chosen are represented in Table 5. Moreover, we maintain the **required_info** entity.

Restaurant	Hotel
pricerange	pricerange
area	area
bookday	bookday
bookpeople	bookpeople
food	type
name	name
booktime	bookstay

Table 5: Mapping of slots from the restaurant domain to the hotel domain

For the mapping of the slots, there are two things that were considered: the mandatory slots, and the slots required for each intent (*find_restaurant* and *book_restaurant*). The mandatory slots for the hotel domain were calculated using the same method as described in Section 2.3. The mandatory slots determined for the hotel agent are **hotel-bookday**, **hotel-bookstay** and **hotel-bookpeople**.

However, there are two slots from the hotel domain that are not considered from the hotel domain: the **internet** and **parking** slots, which could not be correctly labeled in the utterances. Also, we do not consider the slot **stars** because there is no direct correspondent between the restaurant domain and the hotel domain, and it is not mandatory for the functioning of the agent.

We also change the NLU training data. To obtain the training data, we use the approach described in Section 2. Moreover, we adapt each of the Rasa stories by making the mapping of the slots, and select a random value it can take from the annotated utterances in the training data. For example, in the story represented in Listing 1, the first user turn would change the pair (*restaurant-name*, *cafe uno*) to (*hotel-name)*, *allenbell*).

Using this process, we guarantee that the hotel agent recognizes utterances with information regarding the hotel domain, and are able to check if the conversation paths defined by the stories in the restaurant agent can be used for the hotel domain. Concluding, we can now specify the text for the Responses, implementing the Custom Actions and choosing the agent configuration, which is detailed in the repository[6].

4. Evaluation

In this section, we describe how we implement the remaining parts of the conversational agents, and conduct the evaluation of our agents and the results.

4.1. Objectives

There are two objectives regarding this evaluation:

- Prove the quality of the framework developed in order to expedite the creation of Rasa agents.
- Check if it is possible to transfer knowledge between domains.

For the first objective, we perform an evaluation on a restaurant agent created with only domain-specific dialogues. For the second objective, we perform an evaluation on the hotel agent, which is created by transferring knowledge from the restaurant agent, as explained in Subsection 3.2.

For each agent, a two-part evaluation was designed. In the first part of the evaluation, we perform an automatic NLU evaluation provided by Rasa that allows to collect metrics such as Precision, Recall and F1-Score for both the intents and the entities of the agents. These metrics are calculated as a Micro, Macro, and Weighted average, as well as for each intent/entity. Next, we perform a human evaluation. This evaluation is composed by an interaction with the agent's agent, followed by the fill of the questionnaire. During the interaction, some metrics such as the NLU errors, action choice errors, and number of turns required to complete the task are noted. In the form, we collect information about the user experience during the interaction with the agent.

Thus, both the performance of the agent during the interaction as well as the results from the questionnaire will allow us to determine if the interaction with the agent was good, and take conclusions as if the objectives of this evaluation were reached.

4.2. Materials

For the NLU evaluation, we require labeled utterances with both the intents and entities for each. Moreover, for the human evaluation, we need to specify tasks for the user to perform and also to create a questionnaire for the user to fill.

4.2.1 NLU Utterances

To test both the restaurant agent and the hotel mode, we create the test utterances using the methodology described in Section 2 using the MultiWOZ test set. This method allows to have domain-specific utterances that have labeled intents and entities that have been created by humans.

4.2.2 Tasks

For both agents, we create three different tasks for the user to perform. For each of the tasks, we have the following objective: 1) ability to find a restaurant/place to stay and make a booking, 2) get information from a restaurant/place to stay, and 3) change some details of a booking. Taken these objectives, we then select some properties for the en-

[6]https://github.com/Fogoid/KnowledgeTransfer

tities of the agent and describe the task as a single sentence for the user.

4.2.3 Questionnaire

The questionnaire we used to evaluate the agents had three different parts. The first part of the questionnaire retrieves information such as the age, fluency in English and familiarity with conversational agents of the user. For the second part, we focus on measuring the user experience using the UEQ (Schrepp, Hinderks, and Thomaschewski 2017), by analysing the scales of Attractiveness, Perspicuity, Dependability and Stimulation. For last, the third part will measure the quality of the different parts of the system and the system as a whole. For the last part of the questionnaire, we focus on the capabilities of the questionnaire. The capabilities we evaluate are the ability of the agent to understand the user (Understanding Capabilities), as well as if any limitations have been felt by the user (Limitations) . Also, we want to evaluate if the users felt successful in the engagement (Experience), and if the dialogues with the agent felt natural (Flow).

4.3. Automatic NLU Evaluation

As we mentioned previously, we used the labeled test utterances to perform the automatic evaluation using the Rasa testing capabilities. From this evaluation, it was possible to conclude the quality of the NLU module designed, although it presented issues when labeling entities, where some entities were labeled wrongly.

4.4. Human Evaluation

Our sample was composed of 14 participants. From those, 7 (50%) people started by interacting with the restaurant agent, and the remaining 7 (50%) people started by interacting with the hotel agent. The age of the participants ranged from 16 to 28 years old, where 9 (64.29%) participants identified as male and 5 (35.71%) people identified as female. Moreover, the testers evaluated their fluency with a mean of 5.76 of 7 and the overall experience with these type of agents were 3.71 out of 7. In this section, for visualization purposes, we present the results of the questionnaires on a scale of -3 to 3, instead of a scale from 1 to 7. The metrics for the questionnaire results for the two agents are in Table 6.

4.4.1 Restaurant agent Results

During the interaction with the restaurant agent, the users interacted for a total of 279 turns, where the agent had 57 (20.43%) turns with NLU errors and 22 (7.89%) turns with action selection errors. In the first task, the users did an average of 8.21 turns, with 2.38 turns with understanding errors from the NLU and in 0.62 turns the action selected was not correct. For the second task, an average of 4.07 turns were required to complete the task, 0.84 turns of those were understanding errors. Only one tester got an action selection error during this task. Finally, in the third task, the users required an average of 7.68 turns to complete it, where there were 1.15 turns with understanding errors and 1.07 turns where the action selected was not appropriate.

Finally, regarding the opinion of the users, 7 (50%) of them mentioned interpretation errors and 1 (7.14%) mentioned being impacted by the action choice. Moreover, the users said that overall, the conversation was fluid to complete the tasks, and praised the capabilities of the agent.

4.4.2 Hotel agent Results

During the interaction with the agent, there was a total of 331 turns, where 107 (32.33%) turns had understanding issues from the NLU and there were 10 (3.02%) turns where the action selected was not correct. In the first task, there was a mean of 8.21 turns, having 2 turns with NLU errors and 0.3 turns with a action selection errors. In the second task, the number of turns required to terminate the task were 8.07, and there were 3.69 turns with NLU errors. In this task, the action selected was always the best one. For last, in the third task, the users required 7.35 turns to finish it, where 2.54 turns had NLU errors and 0.46 turns where the action selected was not the correct one.

Regarding the opinion of the users, 8 (57.4%) of the users mentioned the errors of the NLU, but praised the agent for being able to help conclude the tasks. Moreover, the users mentioned that they could talk like they talk with another person.

5. Discussion

In this section, we address the problems we faced creating the conversational agents, discuss the results of our evaluation and see how does it affect the objectives of this work.

5.1. Parsing Data From Datasets

To populate dialogue systems that are divided in the three modules (NLU, dialogue manager and NLG), the correct annotation of the dialogues is extremely important to create these type of systems. This could be seen by the analysis of the two datasets (MultiWOZ and Taskmaster-1), that are two datasets with dialogues where the annotations of both differ.

agent	Metric	UEQ				Capabilities			
		Attract.	Persp.	Dep.	Stim.	Und. Cap.	Exp	Lim.	Flow
restaurant	Mean	1.657	1.662	1.515	1.176	1.59	1.92	1.82	1.79
	Var.	0.79	1.51	0.67	0.76	1.15	0.69	0.70	1.27
	Std. Dev	0.891	1.228	0.817	0.874	1.07	0.829	0.834	1.126
	Conf.	0.423	0.584	0.388	0.415	0.509	0.394	0.397	0.535
hotel	Mean	1.569	1.471	1.5	1.441	1.49	1.16	1.57	1.26
	Var.	0.86	1.48	0.91	1.00	0.9	1.16	1.11	1.12
	Std. Dev	0.93	1.215	0.952	0.998	0.951	0.936	1.053	1.059
	Conf.	0.442	0.577	0.453	0.474	0.452	0.445	0.500	0.503

Table 6: Results of mean, variance, standard deviation and confidence with p=0.05 for the two parts of the questionnaire for the two agents

5.1.1 Natural Language Understanding

The Active Intents and Slots of MultiWOZ allow the labeling of a great part of the utterances as a part of an Intent such as the find_restaurant intent or the inform intent. However, this was not possible to do to utterances such as Affirmations, Refusals or Recommendations that are part of a dialogue the dialogue, which blocks the creation of a more complete NLU module. This could be seen during the definition of the custom actions for the agents, where the agent message was designed in order to avoid responses from the user that would affirm or negate.

To address this problem, one possibility is to create in the dataset an Intent annotation, which allows the users to differentiate all the utterances during the development of the conversational agent.

Besides, there are slots where the search for the value was complicated, or in the worse case, were not able to be considered for the agent. This is the case of *hotel-wifi*, since its state value was not found in the utterance. The lack of this slot hinders the creation of conversational agents, diminishing the functionalities of the agent.

One possible solution for this problem is by storing two values during the annotation of slots in the dataset: the state value and the utterance value, which would simplify the search and mapping of the value in the utterance to the state value.

Looking at the results of the automatic evaluation and user evaluation, we can conclude that using the information from the dataset, we are able to create an agent that is able to understand natural language. This is supported not only by the results of the automatic evaluation, but also by the third part of the questionnaires filled by the users.

However, the agent is by no means perfect. This can be seen by the results of the interaction, where there were 7.89% of the turns in the restaurant agent and 32.33% of the turns in the hotel agent had understanding errors. Furthermore, the results of the automatic evaluation also suggest that are improvements to be made. While the turn percentage with NLU errors of the restaurant agent is in concordance with the results of the automatic evaluation. However, in the hotel domain there is a disparity of the two values. There are two possibilities for this happening. First, although unlikely, since we performed the same process to create the training data for the restaurant agent, the utterances from the users in the test differ from the ones used in the training data. Another possibility is the fact that Rasa is not able to distinct correctly between the slots **hotel-bookstay** and **hotel-bookpeople**, since both use a number regex. To solve this last problem, we could create a single regex, and change the training data to be able to specify the type of number, based on the context.

5.1.2 Dialogue Manager

The results of the human evaluation suggest that we were able to create a good base for the dialogue paths. This is supported by the low percentage of turns where the action selected was not the correct one (7.89% of all turns in the restaurant agent). Moreover, the users stated that the agent helped to conclude the tasks and knew what to say to the agent most of the times. This means that using by using Intents, Slots and also the mandatory slots that we computed, it is possible to create the required actions, as well as the stories to define the dialogue manager module.

Just as we mentioned, only 7.89% of the turns had errors while choosing the action to take. We consider that the main reason for this issue is the lack of stories due to the presence of some errors in the dialogues of MultiWOZ. This would lead often to missing Intents on some utterances, which often led to the elimination of the dialogues. One other possibility is the existence of too many dialogues, where two stories that are equal until a certain turn t, take different actions at turn $t+1$. To bridge this problem, we could perform a filter on the dialogues, eliminating any redundancies. One other possibility

would be to create annotations for the agent actions in the dataset, aiding the development process.

However, the results we obtained might not be completely accurate. First, the number of users that interacted with the agent for the evaluation is low. Second, there were no restrictions in the way the users talked, which means it is not possible to access the quality of the stories created

Concluding, we believe that the benefits of using datasets to create a conversational agent outweigh the constraints imposed by some of the annotation properties. As supported by the results of the UEQ part of the questionnaire, using the datasets can create an agent that is good, motivating to use, where the users feel in control the conversation and interact the way they intend, qualities that are required in a conversational agent.

5.2. Knowledge Transfer

Comparing the results of the human evaluation between the hotel and restaurant domains, we consider that the knowledge transfer between domains was a success. In fact, there were less errors concerning the action selection in the hotel agent that were in the domain agent. Furthermore, although the results of the third questionnaire represent a significant diminish of the quality of the flow, we believe that this scale had worse results due to the influence of the errors caused by the NLU module of the agent. Moreover, as the results of the UEQ suggest the agent is still a good exciting product to use, despite all the understanding errors that it presented.

However, we consider that this method of transferring is still extremely limited. First, the only module to be retained is the dialogue manager. This is for obvious reasons, given that the training data for the NLU module is domain-specific. This can be confirmed by our attempt at adding information to the Taskmaster-1 dataset using the MultiWOZ restaurant agent, where even in dialogues with the same domain, the complexity and language of both datasets differed.

Moreover, in order to transfer knowledge, it is mandatory that the tasks are compatible. In our approach, to have compatible tasks, it is obligatory that the domains have the same number of tasks, and the same number of mandatory slots for each correspondent task, which sometimes is not feasible.

6. Conclusions and Future Work

In this work, we propose different methods to expedite the development of conversational agents. We do so using the Rasa Open Source Framework. In a first method, we use the annotations in MultiWOZ and Taskmaster-1 datasets to create a conversational agent. For last, we explore the possibility of adapting an existing conversational agent to new tasks. We perform an automatic and human evaluation on both methods, which show the effectiveness of using datasets to support the creation of new conversational agents, as well as highlighting the possibility of reusing conversational agents.

References

[1] Pawel Budzianowski, Tsung-Hsien Wen, Bo-Hsiang Tseng, Iñigo Casanueva, Stefan Ultes, Osman Ramadan, and Milica Gasic. "MultiWOZ - A Large-Scale Multi-Domain Wizard-of-Oz Dataset for Task-Oriented Dialogue Modelling". In: *CoRR* abs/1810.00278 (2018). arXiv: 1810.00278. URL: http://arxiv.org/abs/1810.00278.

[2] Bill Byrne, Karthik Krishnamoorthi, Chinnadhurai Sankar, Arvind Neelakantan, Daniel Duckworth, Semih Yavuz, Ben Goodrich, Amit Dubey, Kyu-Young Kim, and Andy Cedilnik. "Taskmaster-1: Toward a Realistic and Diverse Dialog Dataset". In: 2019.

[3] Chih-Wen Goo, Guang Gao, Yun-Kai Hsu, Chih-Li Huo, Tsung-Chieh Chen, Keng-Wei Hsu, and Yun-Nung Chen. "Slot-Gated Modeling for Joint Slot Filling and Intent Prediction". In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers).* New Orleans, Louisiana: Association for Computational Linguistics, June 2018, pp. 753–757. DOI: 10.18653/v1/N18-2118. URL: https://www.aclweb.org/anthology/N18-2118 (visited on 12/08/2020).

[4] Javeria Habib, Shuo Zhang, and Krisztian Balog. "IAI MovieBot: A Conversational Movie Recommender System". In: *CoRR* abs/2009.03668 (2020). arXiv: 2009.03668. URL: https://arxiv.org/abs/2009.03668.

[5] Xiujun Li, Yun-Nung Chen, Lihong Li, Jianfeng Gao, and Asli Celikyilmaz. "End-to-End Task-Completion Neural Dialogue Systems". In: *the 8th International Joint Conference on Natural Language Processing.* IJCNLP 2017, Nov. 2017. URL: https://www.microsoft.com/en-us/research/publication/end-end-task-completion-neural-dialogue-systems/.

[6] Martin Schrepp, Andreas Hinderks, and Jörg Thomaschewski. "Construction of a Benchmark for the User Experience Questionnaire (UEQ)." In: *Int. J. Interact. Multim. Artif. Intell.* 4.4 (2017), pp. 40–44.