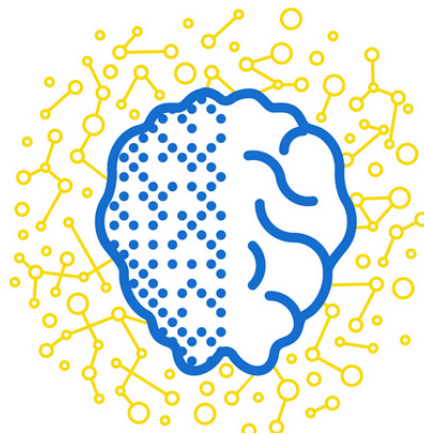


# Implementing a Chatbot using the DeepPavlov framework

EURECOM

BENIGMIM Mohammed Yasser

Supervisors : TRONCY Raphaël, EHRHART Thibault, SCHLEIDER Thomas



Semester Project Report - Spring 2019

Data Science Departement

25/06/2019

## **Abstract**

This is the report of my semester project that we have taken during the Spring semester 2019, and it's about implementing a chatbot using the DeepPavlov framework.

We will begin by presenting an overview of the state of the art concerning the chatbots, and we will see some of them like DBpedia and other ones created by students in EURECOM like Minotour and Doremus. We will therefore explain the functioning of chatbots in general as well as their main characteristics.

The second part will be about presenting the DeepPavlov framework, the way it works, and also one of its main skills which is goal oriented bots, the dataset it works with, and also an important component which is Named Entity Recognition ..

The last part we will be about all the work that we have made during this semester working with DeepPavlov and also the different problems encountered and how we managed to solve them.

Finally, we will end up with a general conclusion of the project as well as the different improvement tracks that could be taken in future work.

# Contents

|          |  |           |
|----------|--|-----------|
| <b>1</b> | <b>Introduction :</b>                                | <b>4</b>  |
| 1.1      | Objectives . . . . .                                 | 5         |
| 1.2      | Methodology . . . . .                                | 5         |
| <b>2</b> | <b>State of the art</b>                              | <b>5</b>  |
| 2.1      | Natural Language processing . . . . .                | 5         |
| 2.2      | Chatbots . . . . .                                   | 7         |
| 2.2.1    | Definition . . . . .                                 | 7         |
| 2.2.2    | How a Chatbot Works . . . . .                        | 7         |
| 2.3      | Overview of Chatbots . . . . .                       | 8         |
| 2.3.1    | DBpedia . . . . .                                    | 8         |
| 2.3.2    | Minotour . . . . .                                   | 10        |
| 2.3.3    | Doremus . . . . .                                    | 11        |
| <b>3</b> | <b>DeepPavlov</b>                                    | <b>13</b> |
| 3.1      | Conceptual overview . . . . .                        | 13        |
| 3.2      | General architecture . . . . .                       | 13        |
| 3.3      | Versioning . . . . .                                 | 14        |
| 3.4      | Named Entity Recognition (NER) . . . . .             | 14        |
| 3.5      | Goal oriented bots . . . . .                         | 15        |
| 3.6      | Datasets . . . . .                                   | 16        |
| 3.6.1    | Dialog State Tracking Challenge 2 (DSTC 2) . . . . . | 16        |
| 3.6.2    | SNIPS dataset . . . . .                              | 17        |
| <b>4</b> | <b>Personal work</b>                                 | <b>18</b> |
| 4.1      | Introduction . . . . .                               | 18        |
| 4.2      | Development environment . . . . .                    | 19        |
| 4.3      | Installation . . . . .                               | 19        |
| 4.4      | Project Management . . . . .                         | 19        |
| 4.5      | Tutorials . . . . .                                  | 20        |
| 4.6      | Tourism chatbot . . . . .                            | 23        |
| 4.6.1    | Objective : . . . . .                                | 23        |
| 4.6.2    | Dataset analysis : . . . . .                         | 23        |
| 4.6.3    | Dataset creation : . . . . .                         | 23        |
| 4.6.4    | problems encountered : . . . . .                     | 26        |
| 4.7      | Intent classification . . . . .                      | 29        |

|          |   |           |
|----------|---|-----------|
| 4.8      | Interaction in the terminal with the go_bot . . . . . | 30        |
| 4.8.1    | The change of the operation system . . . . .          | 30        |
| 4.8.2    | The addition of new data . . . . .                    | 31        |
| <b>5</b> | <b>Conclusion &amp; Future work</b>                   | <b>35</b> |
|          | <b>References</b>                                     | <b>38</b> |
|          | <b>List of Figures</b>                                | <b>40</b> |

# 1 Introduction :

Artificial intelligence is very a general concept but if we want to give it a very simple definition, it can be described as : *"machines or computer programs that learn to perform tasks that require types of intelligence and that are usually performed by humans."*[1]

The Figure 1 below show us the artificial intelligence timeline, where we can see the major steps that AI went through during history.

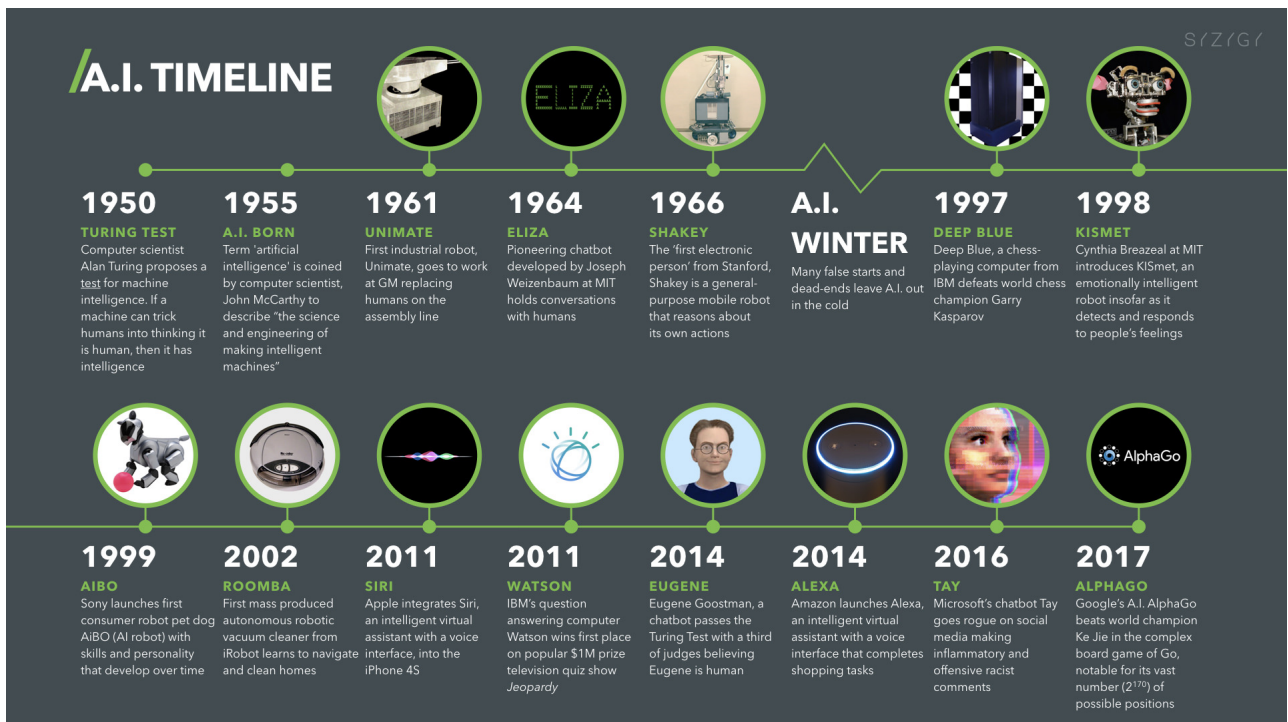


Figure 1. Artificial intelligence timeline <sup>1</sup>

A.I contains lot of subfields like machine learning, deep learning, computer vision, etc ... Chatbots are one of the most important and interesting subfields of A.I, and they have made such a big progress during the last years.

Now everyone has a chatbot in his cell phone with which he can communicate and transmit instructions, and the goal of our project is to build one of them by using an open-source conversational AI library called DeepPavlov[2].

<sup>1</sup><https://digitalwellbeing.org/artificial-intelligence-timeline-infographic-from-eliza-to-tay-and-beyond/>

## 1.1 Objectives

The main objective of the project is to be able to create a Deep Learning-based chatbot specialized for tourism in order to communicate with him and exchange information related to this topic.

The project could be divided into three main stages:

1. Getting started with the framework and the development environment.
2. Understanding and use of the different tutorials available about the DeepPavlov framework.
3. Application of the concept seen previously on our own database.

After building this chatbot, the objective is to move to knowledge graphs and understand all the theory behind it, so that we can transform this new chatbot to a knowledge-graph based one.

## 1.2 Methodology

Throughout this project, we used the official Github page [3] of the framework as well as its documentation [4].

We also asked some questions in the forum [5] which is still in its infancy, and therefore the community behind it is still growing and expanding as well.

The scientific articles and videos allowed us to discover and understand the functioning of several new technical concepts which later were very useful for us to understand in details the functioning of chatbots.

We shared all of our files we worked on throughout the project on our Github repository [6].

## 2 State of the art

In this section, we will discuss about all the documentation part that took place in parallel to the project, and explain how we used all this knowledge to get to the heart of the subject of creating a new chatbot.

### 2.1 Natural Language processing

Natural Language processing (NLP) is one of the most interesting subfields of A.I, and it's a key concept that we need to understand because it's the core behind any conversational AI

and chatbot.

NLP is used to describe the ability of a machine to understand what is said by parsing it, comprehend the meaning and finally determine the appropriate action.

In Wikipedia, the definition is as follows : *"Natural Language Processing is a subfield of Artificial Intelligence that is concerned about the interaction between computers and human natural language"*[7].

There are multiple tasks we find in NLP such as : parsing, stemming, part-of-speech tagging, language detection, etc ...[8]

NLP combines both NLU & NLG.

- **Natural Language Understanding (NLU)** : NLU is a subset of NLP, the main goal of it is to understand the input made of sentences in text or speech format, by identifying the intention of the user (a.k.a *intent*) and the parameters (a.k.a *slots*) of the user sentence.[8]

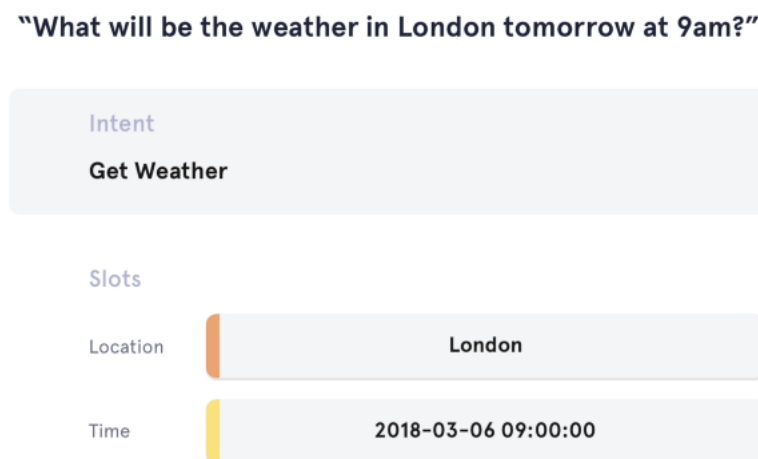


Figure 2. The effect of NLU<sup>1</sup>

The Figure 2 shows us how a Chatbot can understand that the user is looking for the weather in London at 9am, all these information can be understood with NLU.

- **Natural Language Generation (NLG)** : The main goal of this section is to generate a natural language to respond to the user.  
It works like a translator from normal data (*intents, slots*) to a natural language understandable by a human.[8]

---

<sup>1</sup><https://medium.com/snips-ai/an-introduction-to-snips-nlu-the-open-source-library-behind-snips-embedded-voice-platform-b12b1a60a41a>

## 2.2 Chatbots

### 2.2.1 Definition

According to Oxford Dictionaries, a chatbot is : “A computer program designed to simulate conversation with human users, especially over the Internet.”[9]

It’s an assistant that communicates with humans through text messages, it can also simulate a conversation with a user in natural language through messaging applications, websites or mobile apps.

Nowadays, chatbots became much more intelligent and efficient because artificial intelligence which is basically Machine/Deep Learning, NLP and lot of other technologies are integrated now to the functioning of these bots.

Christi Olson, head of evangelism for search at Bing and one of the world experts on chatbots, voice search, and voice assistants said : “Gartner predicts that by 2020 people will have more conversations with chatbots than their spouse.” [10]

### 2.2.2 How a Chatbot Works

If we want to keep things relatively simple we can say that it’s composed of two principal tasks:

- Analysis of the user request : it is about the ability to understand the user’s intent and extract information and relevant information, like in Figure 3. This step is very crucial, because without a good understanding of the user’s purpose, the chatbot can’t give a correct answer.

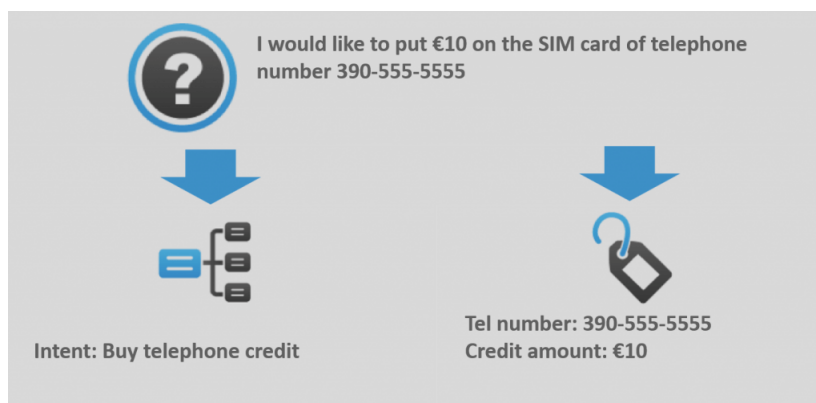


Figure 3. example of intent classification <sup>1</sup>

<sup>1</sup><https://www.expertsystem.com/chatbot>



- Answer generation : The chatbot provides an appropriate response for the user's request, which can be predefined already or contextualized based on the information provided by the user previously, or even retrieved from a backend database.



Figure 4. General overview of the functioning of a Chatbot<sup>1</sup>

The Figure 4 shows the general functioning of a chatbot, from the analysis of the user request to the answer generation.

In the next section we will see some of the chatbots we interacted with during the project and how they helped us to understand better all the mechanisms behind chatbots.

## 2.3 Overview of Chatbots

### 2.3.1 DBpedia

DBpedia in general is a crowd-sourced community effort to extract and create a structured content from all the data coming from Wikipedia, this structure is available on the World Wide Web, and permits to users to semantically query relationships and properties of Wikipedia resources [11] .

They have also created a chatbot that aims to respond to every question by proposing the related Wikipedia or DBpedia page as an answer.

---

<sup>1</sup><https://www.expertsystem.com/chatbot/>

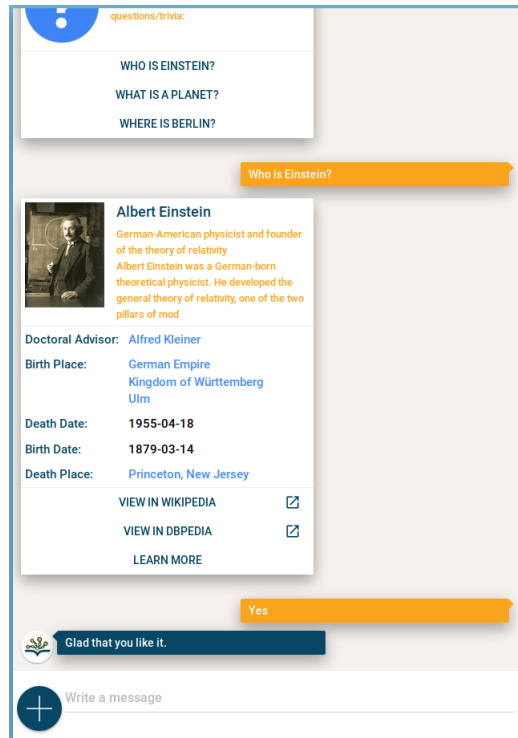


Figure 5. answering "Who is Einstein?"

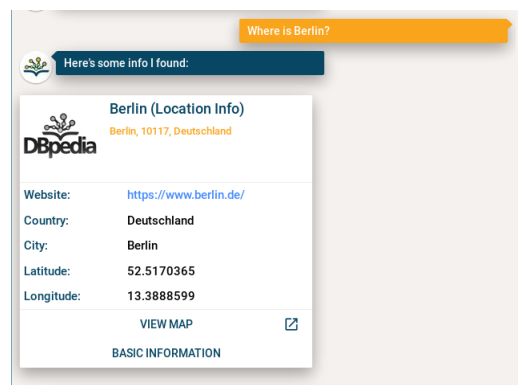
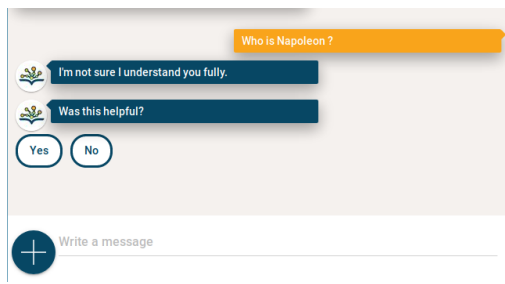


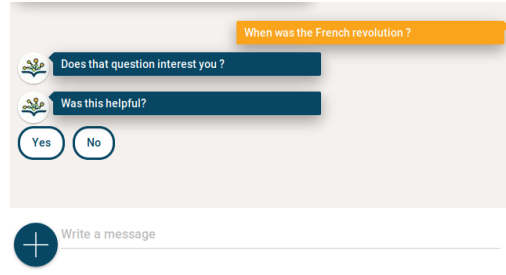
Figure 6. answering "Where is Berlin ?"

The figures above (Figure 14 and Figure 6) show some answers given by the chatbot during the interaction. The user interface is very captivating and also the answers of the chatbots were variate which helps the user to continue interacting with the chatbot.

However, the chatbot is not perfectly working and sometimes it doesn't answer correctly, and we have taking some screenshots showing this in 7a and 7b, where we can see for example that answering to the question "Who is Napoleon ?" was not successful.



(a) answering "Who is Napoleon ?"



(b) answering "When was the French revolution"

Figure 7. Issues faced during their interaction with DBpedia

The idea behind working with DBpedia is to use it as an inspiration for knowledge-graph based chatbots, so that we can have an idea about they work.

### 2.3.2 Minotour

We also interacted with the Minotour Chatbot, a goal oriented bot designed to answer for questions related to tourism .

We tried to interact with Minotour, and as we can see in Figure 8 and Figure 9 it shows that our chatbot is really specific for tourism, and can only answer to user's questions related to events, restaurants and nice places.

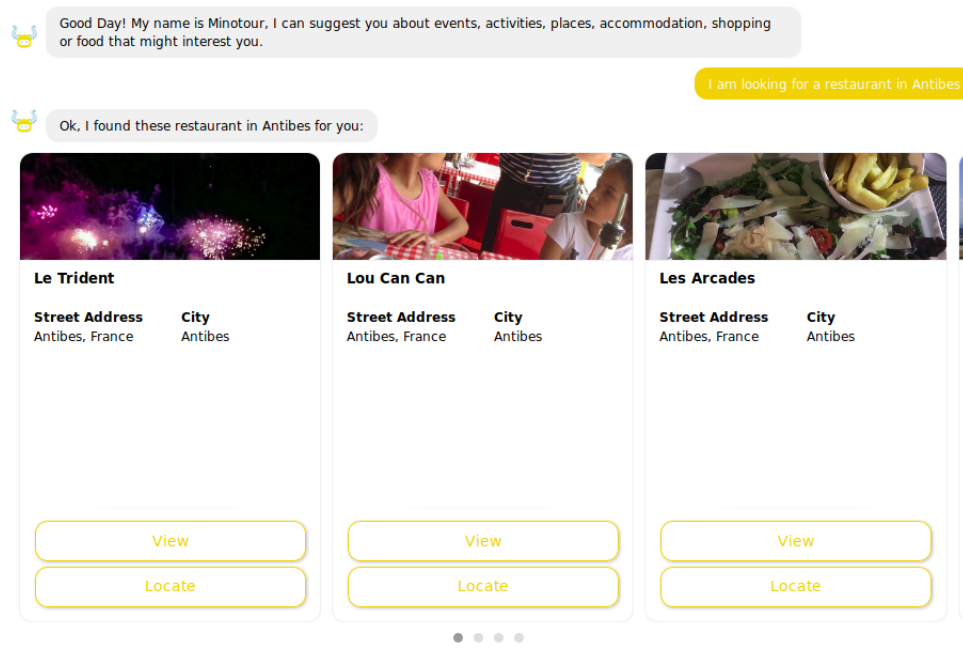


Figure 8. answering "I am looking for a restaurant in Antibes"

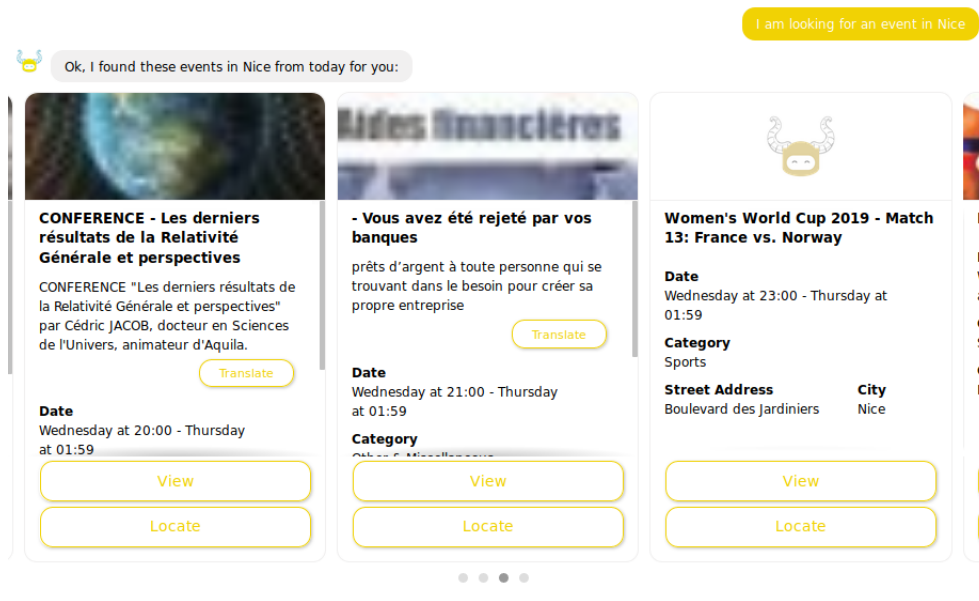


Figure 9. answering "I am looking for an event in Nice"

However, the chatbot was only specific for areas in the south of France, and this is comprehensive because it was developed at EURECOM, this why we had this kind of results in Figure 10

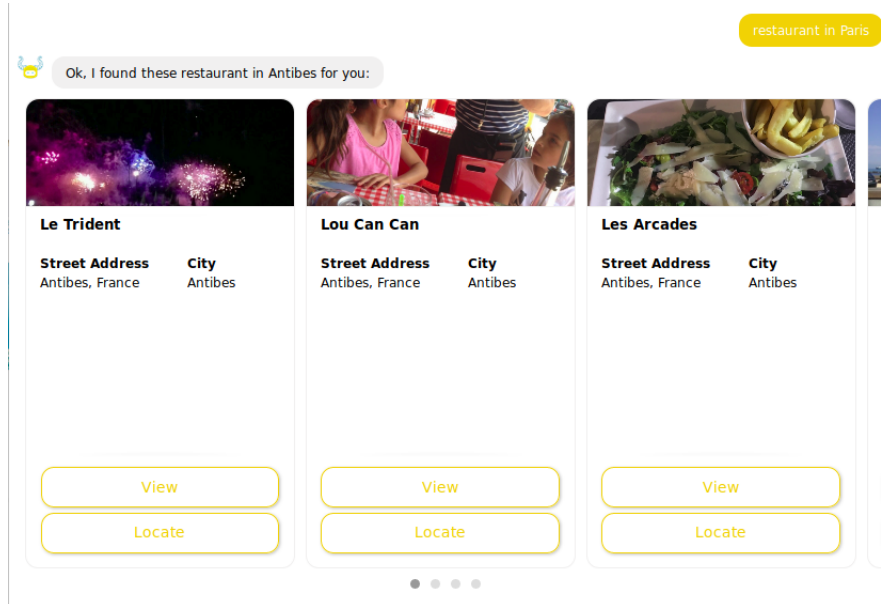


Figure 10. Answering "restaurant in Paris"

### 2.3.3 Doremus

Doremus is also a goal-oriented chatbot, but for answering questions related to classical music.

Doremus was able to give us some answers related to piano pieces and also biographies of some well-known classical artists, and we can see the results in Figure 11 and Figure 12.

The chatbot interface shows a user input 'mozart' in a grey bubble. The chatbot, represented by a circular icon with 'D O R E M U S' and a musical note, responds with a speech bubble containing the following information:

I understood: Mozart

**Leopold Mozart**

This is what I found:

| Born in      | Birth date | Dead in  | Death date |
|--------------|------------|----------|------------|
| South Africa | 1719-11-14 | Salzburg | 1787-01-01 |

**Bio**  
 Johann Georg Leopold Mozart (November 14, 1719 – May 28, 1787) was a German composer, conductor, teacher, and violinist. Mozart is best known today as the father and teacher of Wolfgang Amadeus Mozart, and for his violin textbook Versuch einer gründlichen Violinschule.

Figure 11. answering "Mozart"

The chatbot interface shows a user input 'piano' in a grey bubble. The chatbot responds with two speech bubbles, each containing search results for piano pieces.

**Avant que l'aube ne vienne**

| Composer          | Year | Genre    | Key |
|-------------------|------|----------|-----|
| Jacques Castérède | -    | quartett | -   |

**Comment**  
 Création : Paris (Salle Cotot) 1976, Quatuor Elyséen

**2 sonates**

| Composer         | Year | Genre  | Key |
|------------------|------|--------|-----|
| Rodolfo Halffter | -    | sonate | -   |

**Comment**  
 Réunit : Sonate no 1 (dédicace à Rafael Alberti) ; Sonate no 2 (dédicace à Esteban Salazar Chapela)

Figure 12. Giving some piano pieces

The chatbot was also able to detect the language we used, and switch from English to French and vice versa, as we can see in Figure 15 below.

The chatbot interface shows a user input 'Bonjour' in a grey bubble. The chatbot, represented by a circular icon with 'D O R E M U S' and a musical note, responds with two speech bubbles, one in English and one in French.

Hi! Everyday I try to learn as much as possible about music. Try me if you don't believe it!

Salut! Je suis DOREMUS Bot! Tous les jours j'essaie d'apprendre informations sur la musique classique!

Figure 13. Detection of language

### 3 DeepPavlov

The DeepPavlov framework is an open-source conversational AI library built on TensorFlow<sup>1</sup> and Keras<sup>2</sup> which are both Deep Learning libraries, and the framework has been designed for two things :

- Producing ready chat-bots to use and also some complex conversational systems.
- Natural Language Processing (NLP) and dialog systems research.

This framework has been created by the *Neural Network and Deep Learning Lab* of the *Moscow Institute of Physics and Technology* (MIPT) [12] in a project called *iPavlov* in partnership with a Russian bank called *Sberbank* [13].

DeepPavlov [2] was created in June 2018 with the publication of a scientific paper called *DeepPavlov: Open-Source Library for Dialogue Systems* [14].

#### 3.1 Conceptual overview

The overall goal of DeepPavlov [2] is to permit to AI-application developers and researchers to :

- Work with pretrained NLP models, predefined dialog system components and pipeline templates.
- use the framework to implement and test their own dialog models.
- use the tools provided for application integration with adjacent infrastructure.
- use the benchmarking environment for conversational models and also a uniform access to relevant datasets.

#### 3.2 General architecture



Figure 14. General architecture of DeepPavlov<sup>3</sup>

<sup>1</sup><https://www.tensorflow.org/>

<sup>2</sup><https://keras.io/>

The Figure 14 gives a general overview of the architecture of DeepPavlov.

Let's try to explain these components one by one :

- **Agent**: a conversational agent communicating with users in natural language by text, it can switch between several skills.
- **Skill**: Helps to accomplish one of the user's goal, it can be a simple answer to a question, or the completion of a transaction if it's a ticket reservation for example. Skills must have strings in the input and the output.
- **Component** : it's a reusable part of a skill, and it stands for any kind of function in an NLP pipeline, components can be joined to form a skill.
- **Skill Manager** : performs selection of the skill to generate response, and it can be on different criteria.
- **Chainer** : Helps to build an agent/component pipeline from heterogeneous components (Rule-based/ML/DL), and allows to train and infer models in a pipeline as a whole.

### 3.3 Versioning

DeepPavlov is a framework that was created a year ago , so it's a framework in full construction and in continuous development.

The different versions are available on the official Github page [3], when we took over the project, DeepPavlov was at version 0.2 and at the end of April the framework was upgraded to version 0.3.0.

Currently DeepPavlov is at version 0.3.1.

### 3.4 Named Entity Recognition (NER)

Named Entity Recognition also known as entity extraction is one of the most common tasks in natural language processing (NLP), it helps to classify named entities that are present in a text into categories like "persons", "locations", "organizations", "monetary values" etc...

Here's an example :

|             |   |
|-------------|---|
| <b>text</b> | <i>Michael Jordan of the Chicago Bulls getting a 10-hour documentary on Netflix</i> |
|-------------|---|

---

<sup>3</sup><http://docs.deeppavlov.ai/en/master/intro/overview.html>

Named Entities :

|                     |                       |
|---------------------|-----------------------|
| <b>Person</b>       | <i>Michael Jordan</i> |
| <b>Organization</b> | <i>Chicago Bulls</i>  |
| <b>Organization</b> | <i>Netflix</i>        |

### 3.5 Goal oriented bots

Goal oriented bots help users achieve a predefined goal within a closed domain.

The first step is to understand the user's goal by using natural language understanding, after that when the goal is known, the bot must manage a dialogue to achieve that goal [15].

The Figure 15 below shows the global architecture behind goal oriented bots in DeepPavlov.

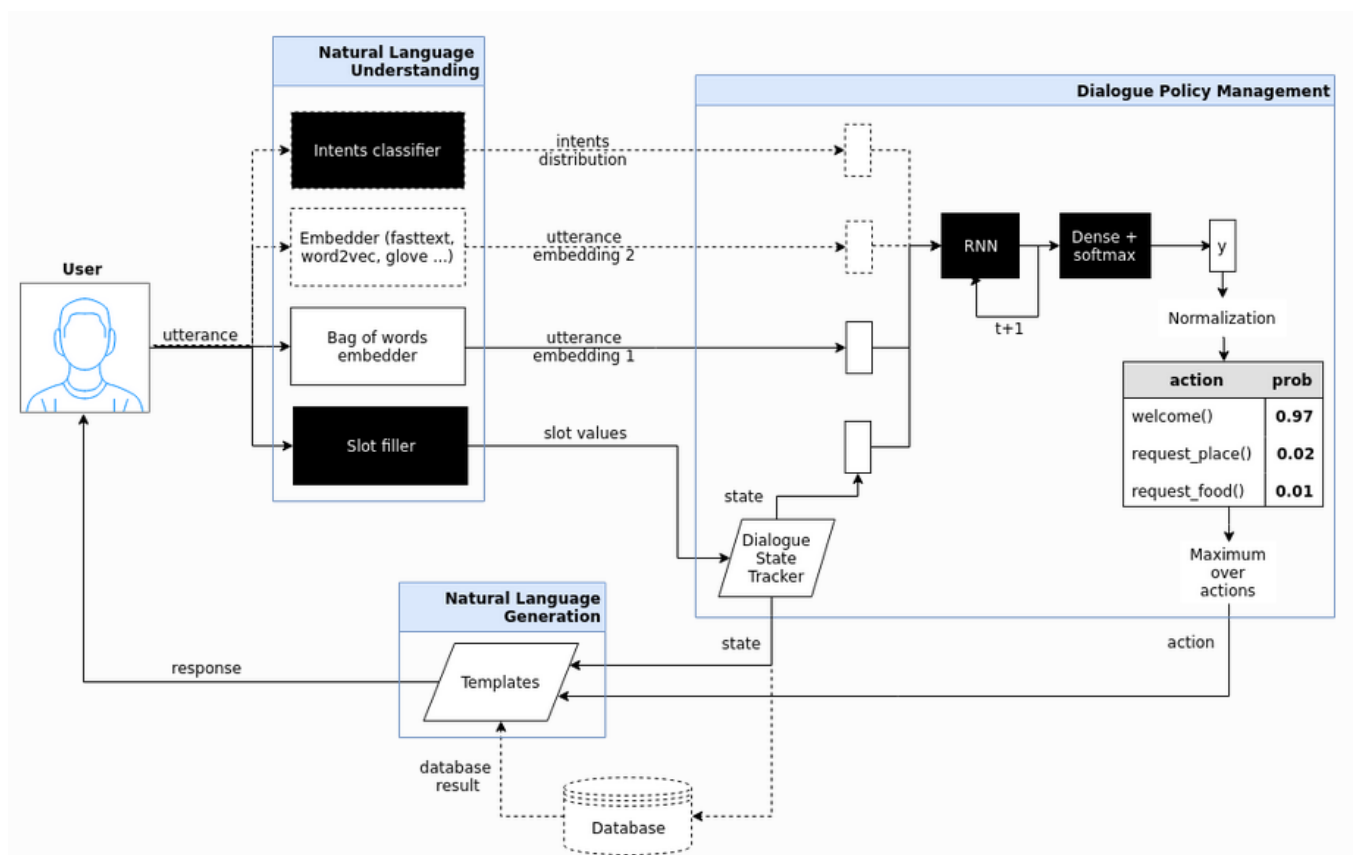


Figure 15. General architecture of goal oriented bots <sup>1</sup>

DeepPavlov is Deep learning based, and we can see that in Figure 15, which use a specific architecture of neural networks called recurrent neural networks to handle the dialogue policy, and therefore the answer to the user's request.

The Figure 16 shows of the interactions we had with the goal oriented bot :

<sup>1</sup>[http://docs.deeppavlov.ai/en/master/skills/go\\_bot.html](http://docs.deeppavlov.ai/en/master/skills/go_bot.html)



- User’s input : it starts with the sign ”x:: ”
- bot’s answer : it starts with the sign ”» ”

```
x::hi
>> What kind of food would you like?
x::I want to eat some italian food
2019-06-14 14:31:52.790 INFO in 'deeppavlov.models.go_bot.network'['network'] at line 472: Made api_call with {'food': 'ita
lian'}, got 13 results.
>> Da vinci pizzeria serves italian food.
x::in which part of town it is ?
2019-06-14 14:32:00.696 INFO in 'deeppavlov.models.go_bot.network'['network'] at line 472: Made api_call with {'food': 'ita
lian'}, got 13 results.
>> Pizza express is a nice place in the centre of town serving tasty italian food.
x::something expensive
2019-06-14 14:32:08.862 INFO in 'deeppavlov.models.go_bot.network'['network'] at line 472: Made api_call with {'food': 'ita
lian', 'pricerange': 'expensive'}, got 4 results.
>> Frankie and bennys serves italian food in the expensive price range.
```

Figure 16. *interaction with goal oriented bot*

DeepPavlov has an entire skill for goal-oriented bots only, and they are trained using the DSTC2 database about restaurant reservation, which we are going to see in the next section.

## 3.6 Datasets

### 3.6.1 Dialog State Tracking Challenge 2 (DSTC 2)

The Dialog State Tracking Challenge<sup>1</sup> is a database that was originally created to serve for a *“research challenge focused on improving the state of the art in tracking the state of spoken dialog systems”*. [16].

In other words, the goal of the challenge is to be able to estimate accurately the user’s goal as a dialog processes, and it needs to be robust to errors in speech recognition for example in order to reduce the ambiguity of the speech.

There was a DSTC 1, DSTC 2 and the latest one is the DSTC 3.

DeepPavlov uses DSTC 2, which is more complicated than DSTC 1 because it includes complicated and dynamic dialog states that changes through the dialog, and it’s about restaurant reservation.

DSTC 2 is composed of three type of files :

- **training/validation/test set** : each file contains hundreds of dialogs between the user and the chatbot about restaurant reservations.

---

<sup>1</sup><http://camdial.org/mh521/dstc/>

```
{
  "speaker": 2, "text": "Hello, welcome to the Cambridge restaurant system. You can ask for restaurants by area, price range or food type. How may I help you?", "dialog_acts": [{"act": "welcomensg", "slots": []}],
  "speaker": 1, "text": "I'm looking for a cheap restaurant in the south part of town", "goals": {"pricerange": "cheap", "area": "south"}, "dialog_acts": [{"slots": [{"pricerange": "cheap"}], "act": "inform"}, {"slots": [{"area": "south"}], "act": "inform"}],
  "speaker": 2, "text": "api_call area='south' food='dontcare' pricerange='cheap'", "dialog_acts": [{"act": "api_call", "slots": [{"pricerange": "cheap"}, {"area": "south"}]}, {"db_result": {"pricerange": "cheap", "area": "south", "addr": "Cambridge Leisure Park Clifton Way", "phone": "01223 327908", "food": "Portuguese", "postcode": "CB1 7DY", "name": "Nandos"}}, {"speaker": 2, "text": "Nandos is a nice place in the south of town and the prices are cheap.", "dialog_acts": [{"act": "inform_area+inform_pricerange+offer_name", "slots": [{"name": "Nandos"}, {"pricerange": "cheap"}, {"area": "south"}]}],
  "speaker": 1, "text": "What is the address", "goals": {"pricerange": "cheap", "area": "south"}, "dialog_acts": [{"slots": [{"slot": "addr"}], "act": "request"}],
  "speaker": 2, "text": "Sure, Nandos is on Cambridge Leisure Park Clifton Way.", "dialog_acts": [{"act": "inform_addr+offer_name", "slots": [{"name": "Nandos"}, {"addr": "Cambridge Leisure Park Clifton Way"}]}],
  "speaker": 1, "text": "Thank you good bye", "goals": {"pricerange": "cheap", "area": "south"}, "dialog_acts": [{"slots": [], "act": "thankyou"}, {"slots": [], "act": "bye"}],
  "speaker": 2, "text": "You are welcome!", "dialog_acts": [{"act": "bye", "slots": []}]}

```

Figure 17. DSTC2 dataset

- **SQLite database** : this database contains a list of restaurants with different attributes (pricerange, phone number, area, etc.. ).

| i   | addr                                      | pricerange | food           | phone        | postcode | area   |
|-----|---|------------|----------------|--------------|----------|--------|
|     | Cambridge Leisure Park Clifton Way ...    | cheap      | chinese        | 01223 244277 | CB1 7DY  | south  |
| 71  | Castle Street City Centre                 | expensive  | indian         | 01223 366668 | CB3 0AH  | west   |
| 290 | Mill Road City Centre                     | expensive  | gastropub      | 01223 247877 | CB1 3NL  | east   |
|     | Cambridge Leisure Park Clifton Way        | cheap      | portuguese     | 01223 327908 | CB1 7DY  | south  |
| 20  | Milton Road Chesterton                    | cheap      | italian        | 01223 351707 | CB4 1JY  | north  |
| 2g  | Cambridge Leisure Park Cherry Hill ...    | expensive  | mexican        | 01223 400170 | CB1 7DY  | south  |
| 17  | Magdalene Street City Centre              | cheap      | vietnamese     | 01223 362456 | CB3 0AF  | west   |
| 169 | High Street Chesterton                    | expensive  | asian oriental | 01223 356555 | CB4 1NL  | north  |
|     | St. Michael's Church Trinity Street Cl... | expensive  | european       | 01223 309147 | CB2 1SU  | centre |

Figure 18. SQLite database

- **template file** : it's a text file that contains a template of answers of the chatbot regarding each combination of entities.

```
bye      You are welcome!
canthear Sorry, I can't hear you.
canthelp_area I'm sorry but there is no #area american restaurant in the #area of town.
canthelp_area_food Sorry there is no #food restaurant in the #area of town.
canthelp_area_food_pricerange Sorry there is no #pricerange restaurant in the #area of town serving #food food.
canthelp_area_pricerange Sorry there is no #pricerange restaurant in the #area of town serving #area american food.
canthelp_food I am sorry but there is no #food restaurant that matches your request.
canthelp_food_pricerange Sorry there is no #food restaurant in the #pricerange price range.
confirm-domain You are looking for a restaurant is that right?
expl-conf_area Did you say you are looking for a restaurant in the #area of town?      Ok, a restaurant in any part of town is that right?
expl-conf_food You are looking for a #food restaurant right?      You are looking for a restaurant serving any kind of food right?
```

Figure 19. Template file

This dataset is going to be used in the goal oriented chatbots that we are going to see more in details later.

### 3.6.2 SNIPS dataset

The SNIPS dataset is a simple database that aims to train the chatbot for intent classification.

The intent means the general purpose of a sentence, in other words the objective of the user's

sentence.

The format is also simple, we have a .csv file with two columns, the first contains a sentence that can potentially be written by any user.

the second column contains the intent that corresponds to this sentence.

For example :

| text  | intent        |
|---|---------------|
| Can you add a track to my Spain Top 50 playlist | AddToPlaylist |
| what is the weather forecast for Maryland       | GetWeather    |
| Give The Hollow Man three out of 6 stars        | RateBook      |

the SNIPS contains around 15 000 rows, which means it's a very rich and varied dataset.

## 4 Personal work

### 4.1 Introduction

In this section, we will review all the work we did during this project, which lasted for a period of three and a half months.

This work was directed by my supervisors on a weekly basis, where the final goal was to create a chatbot related to tourism thanks to the DeepPavlov framework.

DeepPavlov is a growing and continuously expanding library, and this made the project very challenging and interesting at the same time, as I discovered a lot of new concept and especially a new segment of artificial intelligence that is the NLP.

Regarding the rest of this section, we will begin by explaining how we managed our project during this semester and after that how we set up the proper development environment to work with the library on python.

Then we will discuss about the different tutorials that we handled and that were available in version 0.2 and how they allowed us to fully understand the library and chatbots in general. After that, we will start analyzing the database that we have and comparing it to the ones that exist, but especially those that are usable by the Deeppavlov framework, in order to know what are the manipulations to be done to successfully transmit these data to the chatbot for his training.

Finally, we will discuss about intent classification which was a tutorial in the new version 0.3 and also about the last part of the project that focused on our interaction with the chatbot through the terminal, and the results after adding new data to it. To conclude, we will end

up with a conclusion and some instructions to follow that we have set up for all those who will want to continue working on chatbots using DeepPavlov.

## **4.2 Development environment**

### **4.3 Installation**

DeepPavlov is a framework that normally works on Linux, and was later integrated with Windows. It is compatible only with Python 3.6 and does not work with version 3.5.

In the beginning we chose to install DeepPavlov on Windows in an Anaconda environment, because we didn't have Linux installed on our machines, but later in the project we were forced to install Linux because of some problems we will discuss about in details in the upcoming sections.

After downgrading the Python version from 3.7 to 3.6, we had some dependency issues when it came to working with a library called TensorFlow.

After several searches on different forums on the internet, we were able to solve this by modifying the version of a python protocol file called "protobuf" from 3.6.1 to 3.6.0.

After that we were ready to use our DeepPavlov framework on Anaconda without any problem.

The application we used to code throughout the project was jupyter notebook, because the tutorials on the Github page of the framework were jupyter notebook files, but also because it allowed to have code interfaces divided into several cells and to add easily readable comments and results analysis.

### **4.4 Project Management**

The project began on the 8th of March 2018 with a kick-off meeting with our supervisors, where there was a presentation of the project and the objectives to be achieved throughout the semester.

Regarding the organization, we made a presentation every Monday in front of our supervisors where we present our progress during the week that ended and the different issues that we faced during the work.

After presenting the progress, we receive a feedback from supervisors and councils as well as suggestions for improvements for the next week in order to progress and move forward.

The supervisors were also available throughout the week for questions or advice that allowed us to solve problems and keep moving forward.

## 4.5 Tutorials

Before starting the tutorials, we must specify that the majority of those that were available in version 0.2 were removed from the official Github repository during the transition to version 0.3, and therefore we can not unfortunately mention the official source of these tutorials since they are no longer available, even in previous Github versions.

Fortunately, we still saved the older versions and they are present on our own Github repository[6].

- **Hello Bot tutorial** : The goal of this first tutorial, as its name suggests, is to create a simple chatbot based on pattern matching. The tutorial begins with the creation of the chatbot, where the basic idea is to go search for pre-defined patterns in the input text and respond based on the answers already specified.

The patterns are defined in the Figure 21 below:

```
skill_hello = PatternMatchingSkill(['Hello world'], patterns=['hi', 'hello', 'good day'])
skill_bye = PatternMatchingSkill(['Goodbye world', 'See you around'], patterns=['bye', 'chao', 'see you'])
skill_fallback = PatternMatchingSkill(['I don't understand, sorry', 'I can say "Hello world"'])
```

Figure 20. Definition of patterns

If the input text does not match any pattern, the chatbot responds with an answer contained in the **fall\_back** skill.

The Figure 21 shows the answers to a batch of requests made to a chatbot based on pattern matching :

```
HelloBot(['Hello!', 'Boo...', 'chao'])
['Hello world', "I don't understand, sorry", 'Goodbye world']
```

Figure 21. Bot's answers to the batch of requests

At the end of the tutorial, there's a small exercise proposed that we have tried to do, and knowing that the chatbot was basic, we still wanted to test if it was able to detect some typos .

the Figure 22 shows that the chatbot is actually not able to detect this and responds with the skill\_fallback as soon as there is a missing term, in this case it was the question-mark "?" in the sentence *What's your name ?*.

On the other hand, as soon as the complete pattern is detected, the chatbot responds

with the appropriate skills by hardly taking into consideration the characters that surrounds the sentence.

```
#Creation of WhatIsYourName skill
skill_name = PatternMatchingSkill(['My name is Pavlov'], patterns=["What is your name ?", "your name ?",
                                                                    "Do you have a name ?"])
skill_fallback = PatternMatchingSkill(['I don\'t understand, sorry', 'I can say "Hello world"'])

#Creation of the a new agent only for the WhatIsYourName skill
NameBot = DefaultAgent([skill_name,skill_fallback], skills_processor=HighestConfidenceSelector())

NameBot(["What's your name ?"])

['My name is Pavlov']

#One of the limitations of this simple Agent is that he need to find the full pattern
#to be able to do the matching
#in this case the "?" is missing

NameBot(["What's your name "])

['I don't understand, sorry']

#Whenever he finds the full pattern he answers without taking in account the other things
NameBot([" 1234 What's your name ? 5678aeifsjdkzekbj "])

['My name is Pavlov']
```

Figure 22. Tutorial exercise

- **Data preparation tutorial :**

The goal of this tutorial is to get used to the format of the data, and to know how to read the database and then transform it so that it can be read by the machine.

The starting database looks like Figure 23 below :

```
-DOCSTART- -X- -X- 0

EU NNP B-NP B-ORG
rejects VBZ B-VP 0
German JJ B-NP B-MISC
call NN I-NP 0
to TO B-VP 0
boycott VB I-VP 0
British JJ B-NP B-MISC
lamb NN I-NP 0
. . 0 0

Peter NNP B-NP B-PER
Blackburn NNP I-NP I-PER
```

Figure 23. The starting database

In the tutorial we were asked to build a class called `NerDatasetReader` which reads the datasets, and outputs dictionary with fields `train`, `test`, and `valid`.

The Figure 24 shows the class we coded :

```
from pathlib import Path

class NerDatasetReader:
    def read(self, data_path):
        data_parts = ['train', 'valid', 'test']
        extension = '.txt'
        dataset = {}
        for data_part in data_parts:
            file_path = Path(data_path) / Path(data_part + extension)
            dataset[data_part] = self.read_file(str(file_path))
        return dataset

    @staticmethod
    def read_file(file_path):
        Samples = []
        Tokens = []
        Tags = []
        with open(str(file_path), 'r', encoding='utf8') as f:
            for line in f:
                if line.strip():
                    list_line = line.split()
                    if (list_line[0] == "-DOCSTART-"):
                        continue
                    else:
                        Tokens.append(list_line[0])
                        Tags.append(list_line[-1])

                else:
                    Samples.append((Tokens, Tags))
                    Tokens = []
                    Tags = []
            if (len(Tokens) != len(Tags)):
                print("Problem")
        return Samples
```

Figure 24. The class `NerDatasetReader`

This class helped us to read to whole dataset and modify it so that it looks like in Figure 25 :

```
{'train': [(['Mr.', 'Dwag', 'are', 'derping', 'around'], ['B-PER', 'I-PER', 'O', 'O', 'O']),
....],
'valid': [...],
'test': [...]}
```

Figure 25. The output of the class `NerDatasetReader`

## 4.6 Tourism chatbot

### 4.6.1 Objective :

After doing the tutorials and spending time on the documentation, we had to start thinking and working on the new chatbot.

The ultimate goal was to create a goal oriented bot on tourism.

The first step was to know the format of the training data of the goal oriented bot, and how to modify our own data so that it becomes compatible with the goal oriented bot.

### 4.6.2 Dataset analysis :

In order to do this data analysis we looked at the tutorial that corresponded to the goal-oriented bot, and we noticed that the training data came from the DSTC2.

However in the documentation [4], they specified that it is possible to train the chatbot with our personal data provided that it has the form of DSTC2 and we can see that on the Figure 26 below :

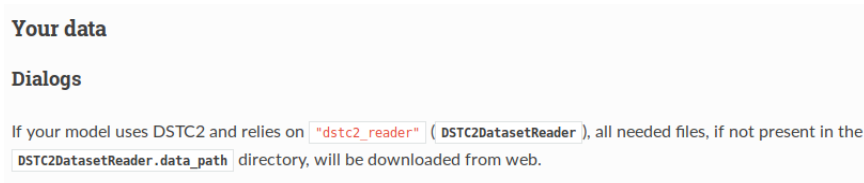


Figure 26. Documentation: using our own data with the go\_bot

After reading this part we concluded that it was possible to train a chatbot with our personal database provided that it takes a form similar to DSTC2.

In the next section we will see how we changed our database so that it can have the closest possible format to DSTC2.

### 4.6.3 Dataset creation :

- RASA dataset :

At first, we wanted to modify the training data from another framework called RASA [17], here is the format of this data below:



```
{
  "intent": "find.accommodation",
  "entities": [
    {
      "start": 15,
      "end": 20,
      "value": "today",
      "entity": "datetime"
    },
    {
      "start": 25,
      "end": 29,
      "value": "Nice",
      "entity": "city"
    }
  ],
  "text": "I want to stay today in Nice"
},
```

Figure 27. RASA training dataset

We can see from Figure 27 that the purpose of this database is to train the model to only detect entities and intents, and it's actually the case because it's a training set for the NLU of RASA.

To read this data, we can notice that the sentence above ("I want to stay today in Nice") has two entities ("Nice" and "today"), and one intent ("find.accommodation").

On the other hand, the DSTC2 in the Figure 29 shows that there's a **dialogue** between the user and the chatbot.

We concluded that it's not possible to modify the existing dataset to have the same format as DSTC2, as we only have sentences without any relation between them, so it's impossible to automate this process.

- **Chat logs :**

After concluding that RASA dataset was not necessarily the best initial database to start with, because the fundamental idea of dialogue was not present, we looked at the chat logs of Minotour chatbot, and the Figure 29 show us how it looks like :

| id      | date      | timestamp | platform | channel                               | user        | intent                | confidence | lang | raw_message            | clean_message               | timestamp_a | response                | rating |
|---------|-----------|-----------|----------|---------------------------------------|-------------|-----------------------|------------|------|------------------------|-----------------------------|-------------|-------------------------|--------|
| 991aba  | 2018-12-0 | 1543655   | webchat  | <a href="https://mir">https://mir</a> | c142f751-43 | Default Welcome Int   | 1          | en   | "Hi"                   | "Hi"                        | 1543655891  | "Greetings! My name is  | 0      |
| 47412b  | 2018-12-0 | 1543775   | webchat  | <a href="https://mir">https://mir</a> | 09e777a6-b7 | Default Welcome Int   | 1          | en   | "Hi"                   | "Hi"                        | 1543775136  | "Good day! Are you bore | 0      |
| 867c07  | 2018-12-0 | 1543827   | webchat  | <a href="https://mir">https://mir</a> | 05f8b15d-d1 | Default Welcome Int   | 1          | en   | "Hi"                   | "Hi"                        | 1543827025  | "Hi! My name is Minotou | 0      |
| 21e410  | 2018-12-0 | 1543831   | webchat  | <a href="https://mir">https://mir</a> | 05f8b15d-d1 | Default Welcome Int   | 1          | en   | "Hi"                   | "Hi"                        | 1543831251  | "Hello! My name is Minc | 0      |
| 45d001  | 2018-12-0 | 1543834   | webchat  | <a href="https://mir">https://mir</a> | b51d1308-04 | Default Welcome Int   | 1          | en   | "Hi"                   | "Hi"                        | 1543834266  | "Good day! Are you bore | 0      |
| f8cd98c | 2018-12-0 | 1543834   | webchat  | <a href="https://mir">https://mir</a> | b51d1308-04 | find.event            | 1          | en   | "could you please give | "could you please give me € | 1543834324  | <result_card>           | 0      |
| 011937  | 2018-12-0 | 1543834   | webchat  | <a href="https://mir">https://mir</a> | b51d1308-04 | find.alternatives     | 0.50999    | en   | "and in Canbnes ?"     | "and in Canbnes ?"          | 1543834340  | <result_card>           | 0      |
| 6384c4  | 2018-12-0 | 1543834   | webchat  | <a href="https://mir">https://mir</a> | b51d1308-04 | find.alternatives     | 0.75124    | en   | "cannes ?"             | "cannes ?"                  | 1543834359  | <result_card>           | 0      |
| ec7e9a  | 2018-12-0 | 1543834   | webchat  | <a href="https://mir">https://mir</a> | b51d1308-04 | Default Fallback Inte | 1          | en   | "monacco"              | "monacco"                   | 1543834413  | "Sorry what was that?"  | 0      |

Figure 28. Minotour Chat logs

The idea of dialogue in this case was present and we made the following manipulations:

1. Keep the columns we need : We only kept the column representing the ID of the dialogue because it was unique for each dialogue, the user message and the intent.
2. Duplicate the intent column : We decided for the sake of simplicity, even if we know that it is not the case, to consider the intent of the user the same as the answer of the bot, because the intent of the bot's answer was present under the name of act in the DSTC2.
3. : Create a .json file : We created a function that creates a .json file which doesn't look exactly the same as the DSTC2 but almost.

The code is shown in the ?? below :

```
import json

def fill_dict(dico,t,i, t_b, a,indice):
    if (indice % 2 ==0) :
        dico["speaker"]=1
    else:
        dico["speaker"]=2
    dico["text"] = t
    dico["intent"] = i
    dico["text_bot"] = t_b
    dico["act"] = a

dialog_id = "9cd817ac-4f43-f5c8-8311-986630b4eb63"
indice = -1
for index,row in concat_data.iterrows():
    dict_line= dict.fromkeys(["speaker","text","intent","text_bot","act"],None)
    fill_dict(dict_line,row['clean_message'],row['intent'],row['response'],row['act'],indice)
    if (row['user'] == dialog_id):
        with open('data.json', 'a') as file:
            indice += 1

            json.dump(dict_line, file,ensure_ascii=False)

            file.write("\n")
    else:
        indice = 0
        dialog_id = row['user']
        with open('data.json', 'a') as file:
            file.write("\n")
            json.dump(dict_line, file,ensure_ascii=False)
            file.write("\n")
```

Figure 29. The function to create the .json file

The .json file was having this format :

```
{"speaker": 1, "text": "find the closest hotel", "intent": "find.accommodation", "text_bot": "What is the city?", "act": "find.accommodation"}
{"speaker": 1, "text": "paris", "intent": "find.accommodation", "text_bot": "What is the city?", "act": "find.accommodation"}
{"speaker": 2, "text": "what city ?", "intent": "find.accommodation", "text_bot": "What is the city?", "act": "find.accommodation"}
```

Figure 30. The .json file

You can find the notebook where we made all these transformations on the Github repository[6].

Knowing that our .json did not really have the same format as the DSTC2, we still decided to continue working with it and to try to see if there was any chance to train the chatbot with these new data.

So we followed the instructions that were on the documentation about using our own data, and more precisely this method:

2. Use `"dialog_iterator"` in dataset iterator config section and `"your_dataset_reader"` in dataset reader config section (recommended):
  - clone `deeppavlov.dataset_readers.dstc2_reader.DSTC2DatasetReader` to `YourDatasetReader` ;
  - register as `"your_dataset_reader"` ;
  - rewrite so that it implements the same interface as the origin. Particularly, `YourDatasetReader.read()` must have the same output as `DSTC2DatasetReader.read()` .
    - `train` – training dialog turns consisting of tuples:
      - first tuple element contains first user's utterance info (as dictionary with the following fields):
        - `text` – utterance string
        - `intents` – list of string intents, associated with user's utterance
        - `db_result` – a database response (optional)
        - `episode_done` – set to `true` , if current utterance is the start of a new dialog, and `false` (or skipped) otherwise (optional)
      - second tuple element contains second user's response info
        - `text` – utterance string
        - `act` – an act, associated with the user's utterance
    - `valid` – validation dialog turns in the same format
    - `test` – test dialog turns in the same format

Figure 31. Method for training with our own data

Unfortunately we had some problems to train the chatbot despite the resemblance with DSTC2, more precisely regarding the reading of the data as well as the iteration on this data

#### 4.6.4 problems encountered :

- **Dataset Reader** : The Dataset Reader is a class that reads the dataset, and since we did not have exactly the same starting point, we modified it and adapted it to the new one, and we called it *TOURSIMDatasetReader*.

We had several problems with the compilation, but after several correction attempts we were able to read the entire dataset but the results were not up to expectations.

The screenshot Figure 32 shows the output of the normal DatasetReader after reading the DSTC2 dataset, and therefore it's what we are supposed to have.

On the other hand Figure 34 shows what we actually had when we iterated over the new dataset we created.

```
({'intents': [{'act': 'inform', 'slots': [['food', 'italian']]}],
  {'act': 'inform', 'slots': [['area', 'west']]}],
  'text': 'im looking for a restaurant in the west part of town serving italian food'},
  {'act': 'request_pricerange',
  'text': 'Would you like something in the cheap, moderate, or expensive price range?'}),
({'intents': [{'act': 'inform', 'slots': [['this', 'dontcare']]}],
  'text': 'doesnt matter'},
  {'act': 'select_pricerange',
  'text': 'Sorry would you like something in the cheap price range or you dont care.'}),
({'intents': [{'act': 'inform', 'slots': [['this', 'dontcare']]}],
  'text': 'i dont care'},
  {'act': 'api_call',
  'text': 'api_call area="west" food="italian" pricerange="dontcare"'}),
  ...)
```

Figure 32. output of DatasetReader (DSTC2)

```
'intents': 'Default Fallback Intent',
  'text': 'how to go from nice airport to downtown ?'},
  {'act': 'find.event', 'text': 'What is the datetime?'}),
({'intents': 'Default Fallback Intent',
  'text': 'where is the tourist office'},
  {'act': 'Default Welcome Intent',
  'text': 'Good day! Are you bored? My name is MinoTour and I can suggest you concert
hat category do you want me to look for?'}),
({'intents': 'Default Fallback Intent',
  'text': 'where is the railway station ?'},
  {'act': 'Default Welcome Intent',
  'text': 'Hi! My name is MinoTour :) What do you want to do?'}),
({'intents': 'find.accommodation', 'text': 'show me an hotel with car park'},
  {'act': 'find.event', 'text': 'What is the datetime?'}),
```

Figure 33. output of TOURSIMDatasetReader

As we can notice the differences are mainly at the level of "slots", and if we look more closely at their content, we note that it is not possible to automate the filling of this list.

There is also the API call to the database, and in our case we do not have it. However, we still wanted to continue working and move to the part of iterating over this data, which we will see in the next section.

### • Dataset Iterator :

In this part, we also had a lot of problems with compilations but after modifying some lines of code while sticking to the method in the Figure 31, we were able to run this section.

the differences were felt much more than before, and the two following figures clearly show it :

```
(([{ 'intents': [], 'prev_resp_act': None, 'text': ''},
  { 'intents': [{ 'act': 'inform', 'slots': [['food', 'traditional']]},
    'prev_resp_act': 'welcomemsg',
    'text': 'traditional'},
  { 'db_result': {},
    'intents': [{ 'act': 'inform', 'slots': [['food', 'traditional']]},
    'prev_resp_act': 'api_call',
    'text': 'traditional'},
  { 'intents': [{ 'act': 'inform', 'slots': [['food', 'italian']]},
    'prev_resp_act': 'canthelp_food',
    'text': 'italian food'},
  { 'intents': [{ 'act': 'inform', 'slots': [['area', 'south']]},
    'prev_resp_act': 'request_area',
    'text': 'south'},
```

Figure 34. output of DatasetIterator (DSTC2)

```
(([{ 'intents': 'Default Fallback Intent',
  'prev_resp_act': None,
  'text': 'what time is it?'}],
[{'intents': 'abandon',
  'prev_resp_act': None,
  'text': 'give me the list of categories'}],
[{'intents': 'abandon', 'prev_resp_act': None, 'text': 'stop'}],
[{'intents': 'Default Welcome Intent',
  'prev_resp_act': None,
  'text': 'Hi minotour'},
{'intents': 'find.activity',
  'prev_resp_act': 'Default Fallback Intent',
  'text': 'i want to go to paris'},
```

Figure 35. output of TOURSIMDatasetIterator

- **Conclusions :**

After this section, we continued to follow the notebook on the goal oriented bot and run the next cells afterwards, but we soon found ourselves stuck.

After several attempts of correction, we stopped because the main reason was because we did not have the same starting form, and therefore we can not hope to arrive at a concrete result.

We concluded that we do need an initial data format that is exactly the same as that of the DSTC2, and that's what was originally mentioned in the documentation but not in a very explicit way.

Our conclusions was clear : *"We cannot train a goal oriented bot with DeepPavlov if we don't have a dataset that has **exactly** the same format as the DSTC2 dataset"*.

However, we continued our work, but this time we headed to the intent classification

which was one of the only notebook that remained after the upgrade to version 3.0.

## 4.7 Intent classification

In the transition to version 3.0, several tutorials were considered as outdated and thus were removed from DeepPavlov’s official Github, the Figure 36 shows it :

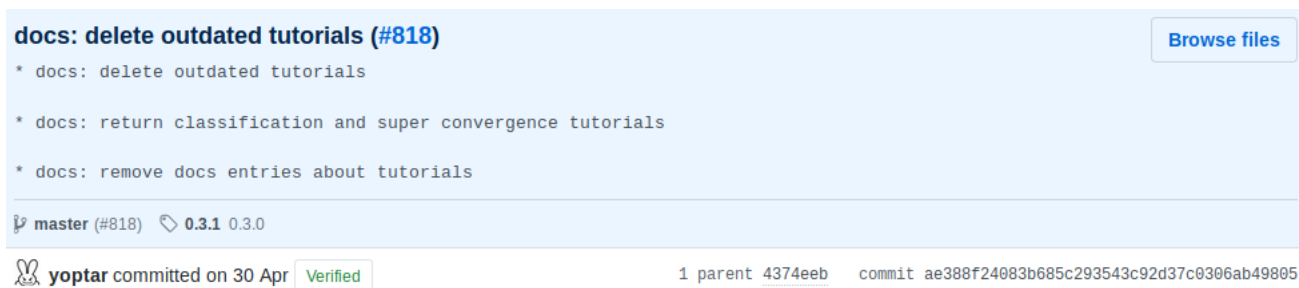


Figure 36. The deletion of tutorials in Github

After the upgrade, we wanted to work on the notebook with the intent classification proposed by the DeepPavlov team which was based on the SNIPS dataset that we discussed earlier, and it’s available on their Github repository [3]

As we have seen, SNIPS is a dataset that allows to train the NLU of a chatbot to recognize the intent in a user’s sentence, which represents somehow the objective of the sentence.

The goal here was not only to understand the notebook’s functioning in general, but also to try and test whether it is possible to do this with our own data.

In this case, our data was always those of Minotour’s chat logs, and since SNIPS dataset contains only two columns with the sentence in one column and the intent corresponding to it in the other, we thought it was possible to modify the chat logs so that they have the exact format of SNIPS dataset.

All of our work on this section can be found in our Github repository.[6] After successfully modifying the chat logs so that they have the same format as SNIPS, we imported it into the notebook on the intent classification that was made by the DeepPavlov team, and we had results this time.

The accuracy of the NLU at the level of the intent classification when it was trained on the chat logs has reached 56%, and it is actually low compared to the case where the training was carried out on the SNIPS dataset, because it was 98% of accuracy, but this was understandable because of the following reasons:

1. **The size of the data** : after grouping all the available chat logs, we had barely managed to have 4159 rows in the dataset.

SNIPS dataset contained around 15,000 lines, so having a low accuracy is quite understandable because the data size is very small.

2. **The variety of data:** Our new dataset, despite its 4159 lines, had only 1900 different lines and so all the rest was actually duplicates, which also explains the bad accuracy, since we found ourselves sometimes with two lines where the sentence is the same but the intent is different, which complicates the training of the NLU.

The results came to support this, because after deleting all the duplicates we went up to an accuracy of 59% with 2000 rows in the dataset.

After doing what we have just mentioned, we wanted to test to add 25% of the SNIPS dataset to the chat logs, and see if we had a better accuracy, and it was the case because the accuracy has risen to 82%.

To conclude, we can say that the size and the variety of the dataset had a huge impact on the accuracy of the intent classification, and the results we had proves it.

The next section will about the last thing we tried to do : try to interact with the goal oriented bot through the terminal and add new data.

## 4.8 Interaction in the terminal with the go\_bot

Our next goal was to be able to interact initially with the goal-oriented bot across the terminal. To do this, we did this on an Anaconda environment installed on Windows

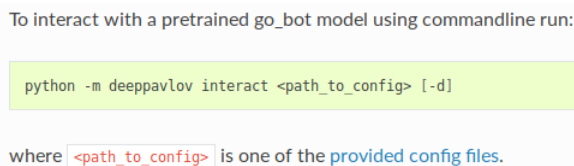
### 4.8.1 The change of the operation system

We tried to interact with the goal oriented bot on Windows, but we had problems with some modules of Visual Studio 2015/2017.

All these problems of dependencies led us to install Linux on the computer, and to test the interaction with the goal-oriented bot.

Indeed the interaction had worked from the first blow, and we were able to test several different discussions with the chatbot who each time interacted in the expected way.

This was done by running the following command that was mentioned in the documentation :



To interact with a pretrained go\_bot model using commandline run:

```
python -m deeppavlov interact <path_to_config> [-d]
```

where <path\_to\_config> is one of the provided config files.

Figure 37. Interaction command with the go\_bot



The Figure 38 shows us an example of this interaction that we took in screenshot :

```
x::I would like to eat in an indian restaurant please
2019-06-16 16:03:26.696 INFO in 'deeppavlov.models.go_bot.network'['network'] at line 472: Made api_call with {'food': 'indian'}, got 22 results.
>> Cocum serves indian food.
x::I want it in the south of the town
2019-06-16 16:03:36.601 INFO in 'deeppavlov.models.go_bot.network'['network'] at line 472: Made api_call with {'food': 'indian', 'area': 'south'}, got 1 results.
>> Taj tandoori is a nice place in the south of town serving tasty indian food.
x::is it cheap ?
2019-06-16 16:03:49.595 INFO in 'deeppavlov.models.go_bot.network'['network'] at line 472: Made api_call with {'food': 'indian', 'area': 'south', 'pricerange': 'cheap'}, got 0 results.
>> What kind of food would you like?
x::expensive then ?
2019-06-16 16:03:55.46 INFO in 'deeppavlov.models.go_bot.network'['network'] at line 472: Made api_call with {'food': 'indian', 'area': 'south', 'pricerange': 'expensive'}, got 1 results.
>> Taj tandoori is a nice place in the south of town serving tasty indian food.
x::Can you give me the adress ?
>> Sure, taj tandoori is on 64 cherry hinton road cherry hinton.
x::and the phone number too
>> The phone number of taj tandoori is 01223 412299.
x::Thanks !
>> You are welcome!
```

Figure 38. Interaction with the go\_bot on Linux

As we can see we tried to find an indian restaurant, and the chatbot was able to help us find it, by following our selection criteria (price = "expensive" , location = "south"), and also by answering to our questions about the address and the number phone too.

We also note the fact that he couldn't find a cheap indian restaurant in the south, because it was not available on the database, and that's why the API call made 0 results when he looked in the dataset.

Now we are going to see how we managed to add new data but also interact on it with the chatbot.

#### 4.8.2 The addition of new data

- **Data with already known entity** : In order to add data, we started by adding restaurants whose origin of food existed on the given base, for example a japanese restaurant.

In the initial database, there was only one japanese restaurant, and we decided to add a second one that differs from the other in the remaining attributes (name, location, pricerange, etc ..).

The Figure 39 shows the new added restaurant which is *Sushi shop* :

| i | addr                    | pricerange | food     | phone        | postcode     | area   | name       |
|---|-------------------------|------------|----------|--------------|--------------|--------|------------|
|   | 36 saint andrews street | expensive  | japanese | 01223 462354 | c.b 2, 3 a.r | centre | wagamama   |
|   | Boulevard Dugommier     | cheap      | japanese | 0678965346   | 06500        | north  | sushi shop |

Figure 39. Adding the new japanese restaurant

Without adding training data on the new restaurant, we tried to interact with the



chatbot and it was a success as we predicted, and the Figure 40 below shows it well.

```
x::Hi I am looking for a japanese restaurant
2019-06-16 16:21:31.668 INFO in 'deeppavlov.models.go_bot.network'['network'] at line 472: Made api_call with {'food': 'japanese'}, got 2 results.
>> Wagamama serves japanese food.
x::I would to eat in a cheap one
2019-06-16 16:22:52.146 INFO in 'deeppavlov.models.go_bot.network'['network'] at line 472: Made api_call with {'food': 'japanese', 'pricerange': 'cheap'}, got 1 results.
>> Sushi shop serves japanese food.
x::Great! In which part of the town it is ?
2019-06-16 16:23:13.3 INFO in 'deeppavlov.models.go_bot.network'['network'] at line 472: Made api_call with {'food': 'japanese', 'pricerange': 'cheap'}, got 1 results.
>> Sushi shop is a nice place in the north of town serving tasty japanese food.
x::Cool! can I have the address please ?
>> Sure, sushi shop is on Boulevard Dugommier.
x::and the phone number please
>> The phone number of sushi shop is 0678965346.
x::Thank you very much
>> You are welcome!
```

Figure 40. Asking the go\_bot about the new japanese restaurant

As we can see above, the chatbot gave us the restaurant already available, but after specifying the price range (*cheap*), the old restaurant did not correspond to what we were looking for, so we fell back on the new restaurant that we had added.

Now, we will see how we instructed the chatbot to go get the restaurants in the new database.

- **Modify the config file**

In Figure 37 we can see that we specify the path to the config file of the go\_bot which is .json file, in order to interact with it.

The Figure 41 shows how the config file looks like :

```

{
  "dataset_reader": {
    "class_name": "dstc2_reader",
    "data_path": "{DOWNLOADS_PATH}/dstc2"
  },
  "dataset_iterator": {
    "class_name": "dialog_iterator"
  },
  "chainer": {
    "in": ["x"],
    "in_y": ["y"],
    "out": ["y_predicted"],
    "pipe": [
      {
        "class_name": "deeppavlov.models.go_bot.wrapper:DialogComponentWrapper",
        "component": { "class_name": "split_tokenizer" },
        "in": ["x"],
        "out": ["x_tokens"]
      },
      {
        "id": "word_vocab",
        "class_name": "simple_vocab",
        "fit_on": ["x_tokens"],
        "save_path": "{MODELS_PATH}/gobot_dstc2/word.dict",
        "load_path": "{MODELS_PATH}/gobot_dstc2/word.dict"
      },
      {
        "id": "restaurant_database",
        "class_name": "sqlite_database",
        "table_name": "mytable",
        "primary_keys": ["name"],
        "save_path": "/home/yasser/Documents/Project/dstc2-test/resto.sqlite"
      }
    ]
  }
}

```

Figure 41. The config file of the go\_bot

We can see that we specified the "save\_path" of the "restaurant\_database" to a local path, which means that we instruct to go\_bot to go retrieve the data from the modified database with the new restaurant added, and this is what happened, and the following message in the terminal shows it ::

```

2019-06-16 22:34:57.927 INFO in 'deeppavlov.core.data.sqlite_database' ['sqlite_database'] at line 63: Loading database from
/home/yasser/Documents/Project/dstc2-test/resto.sqlite.

```

Figure 42. Retrieval of data from the modified database

After having seen how we modified the .json config file, we will see the reaction of the chatbot when we added a restaurant whose culinary origin has never been described before in the SQLite database.

- **Data with unknown entity :**

In this case, we tried to add a new restaurant, and it is an argentinian restaurant.

In the initial database, no argentinian restaurant existed before, and no mention of it in the datasets for training , therefore the entity "argentinian" is new for the chatbot.

The reaction of the chatbot was understandable as we can see in the Figure 43 :

```

x::Hi I want argentinian food
2019-06-16 16:30:41.886 INFO in 'deeppavlov.models.go_bot.network'['network'] at line 472: Made api_call with {'food': 'ind
ian'}, got 22 results.
>> Cocum serves indian food.
x::I said argentinian
>> Cocum serves indian food.
x::a tuscan restaurant ?
2019-06-16 16:31:18.907 INFO in 'deeppavlov.models.go_bot.network'['network'] at line 472: Made api_call with {'food': 'tus
can'}, got 0 results.
>> I am sorry but there is no tuscan restaurant that matches your request.

```

Figure 43. Asking the go\_bot about the argentinian restaurant

As we can see, the word "argentinian" was understood as being "indian" and that's why we received answers about indian restaurants.

On the other hand we can see the word "Tuscan" was understood and verified that there was no "Tuscan" restaurant in the database, but that it was actually cited in the training data, so the chatbot managed to understand the word "Tuscan".

We can therefore conclude that to interact with a chatbot about a restaurant whose culinary origin has never been mentioned before, and so that he can understand the word, it must be entered in the training set in a dialogue form of the same format as the DSTC2.

We tried to add a new dialogue about the Argentinian restaurant, and train our model on it, but we had some problems.

- **Problems encountered :**

After adding the new dialogue on the training/validation/test sets, we ran this command that we have found on the documentation :

You can also train your own model by running:

```
python -m deeppavlov train <path_to_config> [-d]
```

Figure 44. training command for the go\_bot

We made sure the chatbot was reading the changed data with the following message :

```

2019-06-16 18:15:08.162 INFO in 'deeppavlov.dataset_readers.dstc2_reader'['dstc2_reader'] at line 112: [loading dialogs fro
m /home/yasser/Documents/Project/dstc2-test/dstc2-trn.jsonlist]
2019-06-16 18:15:08.462 INFO in 'deeppavlov.dataset_readers.dstc2_reader'['dstc2_reader'] at line 112: [loading dialogs fro
m /home/yasser/Documents/Project/dstc2-test/dstc2-val.jsonlist]
2019-06-16 18:15:08.676 INFO in 'deeppavlov.dataset_readers.dstc2_reader'['dstc2_reader'] at line 112: [loading dialogs fro
m /home/yasser/Documents/Project/dstc2-test/dstc2-tst.jsonlist]

```

Figure 45. The go\_bot reading from the modified dataset

However, in the end we had the following error messages :

```

deeppavlov.core.common.errors.ConfigError: 'obs_size' parameter must be equal to saved model parameter value '611', but is
equal to '612'

```

Figure 46. first error message in the terminal

After modifying in a python file the variable *obs\_size*, we had another error :

```
InvalidArgumentError (see above for traceback): Restoring from checkpoint failed. This is most likely due to a mismatch between the current graph and the graph from the checkpoint. Please ensure that you have not altered the graph expected based on the checkpoint. Original error:
Assign requires shapes of both tensors to match. lhs shape= [612,160] rhs shape= [611,160]
[[Node: save/Assign_1 = Assign[T=DT_FLOAT, class=["loc:@dense/kernel"], use_locking=true, validate_shape=true, _device="/job:localhost/replica:0/task:0/device:CPU:0"](dense/kernel, save/RestoreV2:1)]]
```

Figure 47. second error message in the terminal

We are pretty sure that these two errors are linked, and the searches are in progress to find a solution for it.

## 5 Conclusion & Future work

In this report, we have been able to deal with the entire project from all angles, starting with documentation that started with the launch of the project and continued parallel to it, where we could understand many new concepts that were not easy to tackle the first time and that we used to move forward throughout the project.

Then we started to get a little deeper into the subject by presenting this new framework that is DeepPavlov, explaining how it works and the different components that make it up and that we used in our project.

Finally, we took the time to explain step by step our personal work that extended throughout the semester and through which we learned a lot and discovered a new facet of artificial intelligence that is the NLP and more precisely the world of chatbots.

We had problems with the framework, and this was because it was new and therefore still in full development, but it made the project even more challenging and pushed us to look more to find solutions to the problems encountered.

Regarding the work that can be done in the future, we saw at the end of the previous section, that the interaction with the chatbot could not be executed when we added a discussion in the training set, we have also deleted a random discussion to return to the initial number of discussion and it was a failure too.

We think that some variables have to be modified so that it can work, and that it is very likely that this is only a purely technical problem that can be solved by manipulating the python code inside.

The other skills that constitute Deep Pavlov have not been reviewed in this project, but they are also very interesting, like the Sequence-to-Sequence for example and also the Open-domain answering question.

DeepPavlov is a powerful framework that integrates the network of neurons in its operation, and the results displayed in several tests of chatbots are very encouraging for the rest of this

framework, which promises to be one of the pillars of the advent of chatbots, which are only at their beginning.

## **Acknowledgment**

I would like to extend my gratitude to a number of people whose help was very valuable in this semester project. I would like to thank my supervisors Mr. TRONCY Raphael, Mr. EHRHART Thibault and Mr. SCHLEIDER Thomas for their meaningful assistance, tireless guidance and patience, and also for reviewing this report in draft form. I would also like to thank the administration of Ecole des Mines de Saint Etienne for letting me study a second semester at EURECOM, and to thank EURECOM for accepting me to do it too.

## References

- [1] Definition of artificial intelligence. <https://medium.com/future-today/understanding-artificial-intelligence-f800b51c767f>.
- [2] Neural Networks and MIPT Deep Learning lab. *DeepPavlov*. Moscow Institute of Physics and Technology, 2018. <https://deeppavlov.ai/>.
- [3] Deeppavlov github page. <https://github.com/deepmipt/DeepPavlov>.
- [4] Deeppavlov documentation. <http://docs.deeppavlov.ai/en/master/>.
- [5] Deeppavlov forum. <https://forum.ipavlov.ai/>.
- [6] Github repository of the project. <https://github.com/yasserben/Chatbot-Project>.
- [7] Definition of nlp in wikipedia. [https://en.wikipedia.org/wiki/Natural\\_language\\_processing](https://en.wikipedia.org/wiki/Natural_language_processing).
- [8] Rajai Nuseibeh. Nlp; nlu and nlg conversational process automation chatbots explained. 2018. <https://medium.com/botique-ai/nlp-nlu-and-nlg-chatbots-explained-64820bc32ad>.
- [9] Definition of chatbot in oxford dictionary. <https://en.oxforddictionaries.com/definition/chatbot>.
- [10] Citation about chatbots. <https://www.searchenginejournal.com/future-of-chatbots/278595/close>.
- [11] Definition of dbpedia in wikipedia. <https://en.wikipedia.org/wiki/DBpedia>.
- [12] Moscow institute of physics and technology. <https://mipt.ru/english/>.
- [13] Sberbank of russia. <https://www.sberbank.ru/>.
- [14] Mikhail Burtsev, Alexander Seliverstov, Rafael Airapetyan, Dilyara Baymurzina, Mikhail Arkhipov, Nickolay Bushkov, Olga Gureenkova, Taras Khakhulin, Yuri Kuratov, Denis Kuznetsov, Alexey Litinsky, Varvara Logacheva, Alexey Lyamar, Valentin Malykh,

- Maxim Petrov, Vadim Polulyakh, Leonid Pugachev, Alexey Sorokin, Maria Vikhreva, and Marat Zaynutdinov. Deeppavlov: Open-source library for dialogue systems. 2018. <https://www.aclweb.org/anthology/P18-4021>.
- [15] Andreea Hossmann Michael Baeriswyl Vladimir Ilievski, Claudiu Musat. Goal-oriented chatbot dialog management bootstrapping with transfer learning. 2018. <https://arxiv.org/abs/1802.00500>.
- [16] Dialog state tracking challenge 2 official website. <http://camdial.org/mh521/dstc/>.
- [17] Rasa framework. <https://rasa.com/>.



## List of Figures

|    |  |    |
|----|--|----|
| 1  | Artificial intelligence timeline . . . . .                     | 4  |
| 2  | The effect of NLU . . . . .                                    | 6  |
| 3  | example of intent classification . . . . .                     | 7  |
| 4  | General overview of the functioning of a Chatbot . . . . .     | 8  |
| 5  | answering "Who is Einstein?" . . . . .                         | 9  |
| 6  | answering "Where is Berlin ?" . . . . .                        | 9  |
| 7  | Issues faced during thei nteraction with DBpedia . . . . .     | 10 |
| 8  | answering "I am looking for a restaurant in Antibes" . . . . . | 10 |
| 9  | answering "I am looking for an event in Nice" . . . . .        | 11 |
| 10 | Answering "restaurant in Paris" . . . . .                      | 11 |
| 11 | answering "Mozart" . . . . .                                   | 12 |
| 12 | Giving some piano pieces . . . . .                             | 12 |
| 13 | Detection of language . . . . .                                | 12 |
| 14 | General architecture of DeepPavlov . . . . .                   | 13 |
| 15 | General architecture of goal oriented bots . . . . .           | 15 |
| 16 | interaction with goal oriented bot . . . . .                   | 16 |
| 17 | DSTC2 dataset . . . . .  | 17 |
| 18 | SQLite database . . . . .                                      | 17 |
| 19 | Template file . . . . .  | 17 |
| 20 | Definition of patterns . . . . .                               | 20 |
| 21 | Bot's answers to the batch of requests . . . . .               | 20 |
| 22 | Tutorial exercise . . . . .                                    | 21 |
| 23 | The starting database . . . . .                                | 21 |
| 24 | The class NerDatasetReader . . . . .                           | 22 |
| 25 | The output of the class NerDatasetReader . . . . .             | 22 |
| 26 | Documentation: using our own data with the go_bot . . . . .    | 23 |

|    |   |    |
|----|---|----|
| 27 | RASA training dataset . . . . .                               | 24 |
| 28 | Minotour Chat logs . . . . .                                  | 24 |
| 29 | The function to create the .json file . . . . .               | 25 |
| 30 | The .json file . . . . .                                      | 25 |
| 31 | Method for training with our own data . . . . .               | 26 |
| 32 | output of DatasetReader (DSTC2) . . . . .                     | 27 |
| 33 | output of TOURSIMDatasetReader . . . . .                      | 27 |
| 34 | output of DatasetIterator (DSTC2) . . . . .                   | 28 |
| 35 | output of TOURSIMDatasetIterator . . . . .                    | 28 |
| 36 | The deletion of tutorials in Github . . . . .                 | 29 |
| 37 | Interaction command with the go_bot . . . . .                 | 30 |
| 38 | Interaction with the go_bot on Linux . . . . .                | 31 |
| 39 | Adding the new japanese restaurant . . . . .                  | 31 |
| 40 | Asking the go_bot about the new japanese restaurant . . . . . | 32 |
| 41 | The config file of the go_bot . . . . .                       | 33 |
| 42 | Retrieval of data from the modified database . . . . .        | 33 |
| 43 | Asking the go_bot about the argentinian restaurant . . . . .  | 34 |
| 44 | training command for the go_bot . . . . .                     | 34 |
| 45 | The go_bot reading from the modified dataset . . . . .        | 34 |
| 46 | first error message in the terminal . . . . .                 | 34 |
| 47 | second error message in the terminal . . . . .                | 35 |