Chatbot Project Final Report
Summer 2019

Chatbot With NYTimes API
Implemented by Rasa Stack and Telegram

Ke Ma

Faculty Advisor:
Dr. Fan Zhang

Contents

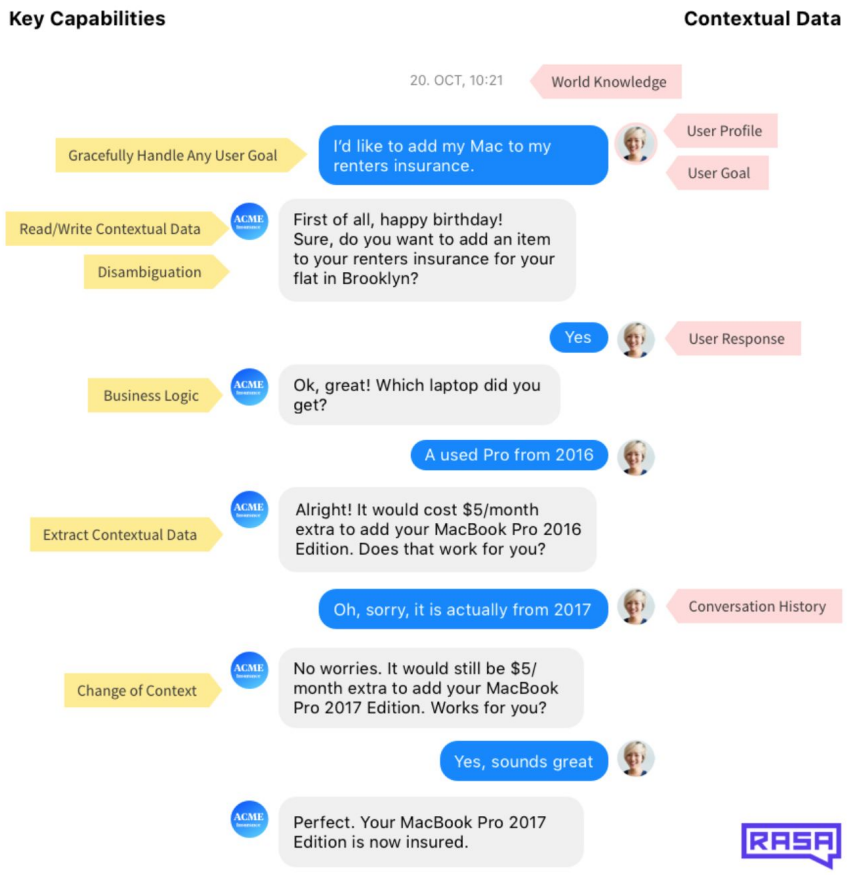
Introduction

Nowadays, the conversational AI system is becoming an important and popular part of our society. Some well-developed examples of conversational AI include Siri from Apple, Alexa from Amazon and Cortana from Microsoft. A conversational chatbot can understand the context of the conversation and can handle any user's intent or goal and help accomplish it as best as possible. This project builds a conversational chatbot about the latest news, books and movies from the New York Times, an American newspaper based in New York City with worldwide influence and readership. The core techniques are Rasa Stack and spaCy language model. The chatbot is able to have conversations with users in three areas: news, books and movie reviews.

Background Information

SpaCy is the language model is going to be used to parse incoming text messages and extract the necessary information. This project uses

en_core_web_md which is English multi-task CNN trained on OntoNotes, with GloVe vectors trained on Common Crawl. Assigns word vectors, context-specific token vectors, POS tags, dependency parse and named entities.



Rasa Stack is a set of open source NLP tools focused primarily on chatbots and AI assistants. In fact, it's one of the most effective and time efficient tools to build complex chatbots in minutes. The chatbot is based on machine

learning model which trained on some example conversations. It consists of two frameworks:

Rasa NLU: a library for natural language understanding (NLU). It performs the duties of intent classification and entity extraction which helps the chatbot to understand the user's input.

Rasa Core: a chatbot framework with machine learning-based dialogue management that predicts the next best action using a probabilistic model like LSTM neural network. The predictions are based on the structured input from the NLU.

Installation & Setup

The project needs a Rasa NLU, Rasa Core and a spaCy language model.

Install latest Rasa NLU

$ pip install rasa_nlu

or $ Python3 install rasa_nlu (https://rasa.com/docs/nlu/installation/)

Install latest Rasa Core

$ pip install rasa_core_sdk

(https://rasa.com/docs/core/installation/)

spaCy+sklearn (pipeline)

$ pip install rasa_nlu[spacy]

$ python3 -m spacy download en

$ python3 -m spacy download en_core_web_md

$ python3 -m spacy link en_core_web_md en

Tensorflow (pipeline)

$ pip install rasa_nlu[tensorflow]

Training And Testing rasa_nlu

The training data is made by a list of messages which expects to receive from the bot. This data is annotated with the intent and entities and Rasa NLU is able to learn to extract. Following are the concepts and examples of intent and entities:

Intent is what the user is aiming for. For instance : when the user asks: *"Could you find a 5-star hotel in downtown LA for me?"* , the intent of this

sentence can be classified as find_hotels which is aiming for finding the hotels.

Entity is to extract the useful information from the user input. From the example above "*Could you find a 5-star hotel in downtown LA for me?*" the entities extracted would be hotel rating and location. Hotel rating — 5 stars and Location — downtown LA.

Below is a brief sample from the training data. You can also add some spelling mistakes or slangs since that will give a flavour of the spoken language to the bot. More the data better the bot would get trained.

```
## intent:greet
- hey
- hello
- hi
- Hi

## intent:fine_ask
- I am good, how are you doing?
- I'm fine, how are you?
- I'm good, how are you?

## intent:fine_normal
- I am doing great
- I'm doing great
- I'm fine
- I'm good

## intent:thanks
- Thanks
- Thank you so much
- Perfect! Thank you
- Thank you
```

Now as our NLU data and pipeline (pipeline: "spacy_sklearn") are ready, it is time to train the bot. We can do so either by running the script in terminal or we can create a python file and run it.

```
python -m rasa_nlu.train -c nlu/nlu_config.yml --data nlu/nlu_data.md -o models --fixed_model_name nlu --project current --verbose
```

This command trains the rasa_nlu using a config file specified by "-c" and data specified by "-data" and stores the model at the directory specified by "-o".

To test model, run `python3 nlu_model.py`. This will train and test the NLU model and save it at 'models/current/nlu'. The same path is passed to NLU Interpreter to parse some sample intents from the user to see if NLU is able to classify the intent and extract the entities correctly.

Training And Testing rasa_core

Our chatbot is now capable of understanding what the user is saying or what the user wants to express. Now it's time to build a dialogue management for bot to respond to the messages. In this part we will teach the chatbot to make

responses by training a dialogue management model using Rasa Core. For dialog training, Rasa core has two main components — domain.yml and stories.md

domain.yml — The domain defines the universe your bot lives in which also include what user inputs it should expect to get, what actions it should be able to predict, how to respond and what information to store.

stories.md — Rasa Core models learn from real conversational data in the form of training stories.md. Stories are real pieces of conversation between users and bots. In each story, user inputs are expressed as intents and the responses of the bot are expressed as actions.

```
$ python -m rasa_core.train -d core/domain.yml -s core/stories.md -o
models/current/dialogue
```

this command trains the rasa_core using a domain file specified by “-d” and stories by “-s” and stores the model at the directory specified by “-o”.

Custom Actions

In training nlu section we defined the utter action by adding an utterance template to the domain file. If we need to run some other code or some third party api, then we would need custom actions. You can create an action server in node.js, Java, or any other language and define your actions there. Here we're using python and rasa does provide 'rasa core sdk" to simplify the work. Rasa Core calls an endpoint specified by us when a custom action is predicted. This endpoint should be a web server that reacts to this call, runs the code and optionally returns information to modify the dialogue state.

to test model start the action server

```
$ python -m rasa_core_sdk.endpoint --actions actions
```

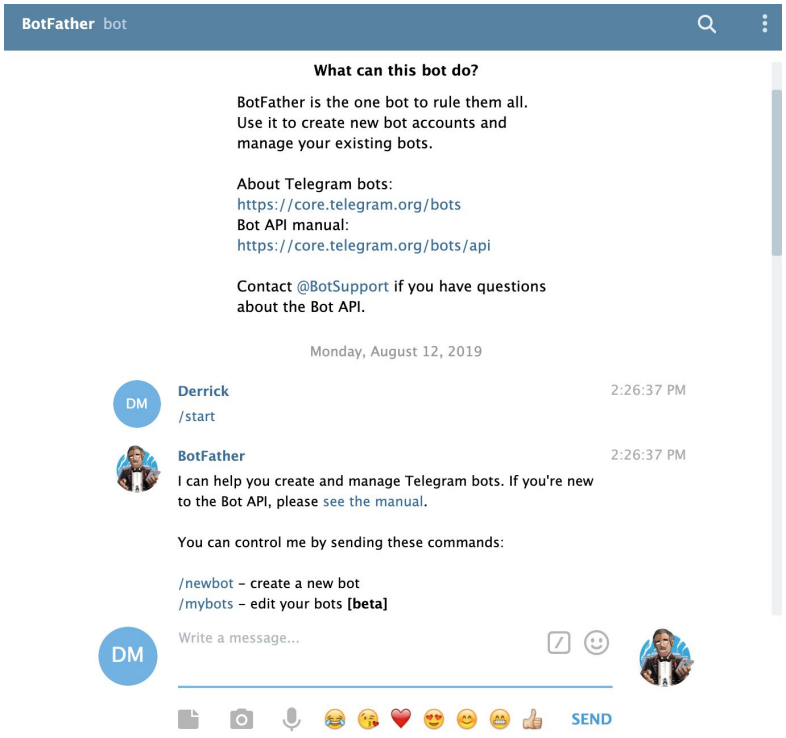
then run the following command to start chat with the bot using terminal

```
python3 -m rasa_core.run -d models/current/dialogue -u models/current/nlu
--endpoints endpoints.yml
```

Telegram Set Up

Now the chatbot needs some interface to chat with the user and for that we chose Telegram bots to implemented. The Bots are third-party applications that run inside Telegram. Users can interact with bots by sending them messages, commands and inline requests. You control your bots using HTTPS requests to telegram bot API.

step 1: create a bot using botfather and get the access token.



step 2: setup a webhook using this URL:

https://api.telegram.org/bot{token}/setWebhook?url={webhook url}

Note:

*token needs to be replaced by your bot token which comes from the botfather

webhook url needs to be https. If you are following along on local machine you can use **ngrok or any other similar service to create a tunnel.

Step 3: create a python file main.py

- import all the dependencies

```
import requests
import json
from flask import Flask
from flask import request
from flask import Response
from rasa_core.agent import Agent
from rasa_core.interpreter import RasaNLUInterpreter
from rasa_core.utils import EndpointConfig
```

- load trained **nlu** model

```
# load trained models
interpreter = RasaNLUInterpreter('./models/current/nlu')
agent = Agent.load('./models/current/dialogue', interpreter=interpreter,action_endpoint=EndpointConfig
```

- function to handle post request from the telegram

```
# accept telegram messages
@app.route('/',methods=['POST','GET'])
def index():
    if(request.method == 'POST'):
        msg = request.get_json()
        chat_id , message = parse_msg(msg)
        response_messages = applyAi(message)
        send_message(chat_id,response_messages)
        return Response('ok',status=200)
    else:
        return '<h1>HELLO</h1>'
```

- help functions

```
# helper function to extract chat id and text
def parse_msg(message):
    chat_id = message['message']['chat']['id']
    txt = message['message']['text']
    return chat_id,txt

# helper function to send message
def send_message(chat_id,messages=[]):
    url = 'https://api.telegram.org/bot'+token+'/sendMessage'
    if messages:
        for message in messages:
            payload = {'chat_id' : chat_id,'text' : message}
            requests.post(url,json=payload)
    return True
```

- Function which interact with the model

```
# get response using rasa
def applyAi(message):
    responses = agent.handle_message(message)
    text = []
    if responses:
        for response in responses:
            text.append(response["text"])
    return text
```

Step 4: time to chat

Now it's time to run the code and chat with bot over telegram!

```
$ python3 main.py
```

Conclusion

From this project, I learned the concept and knowledge about RASA, how to set it up, the difference between rasa_nlu and rasa_core, how to train and test

rasa_nlu and rasa_core models. From the developed chatbot, the project also includes the implementation of the NYTimes API and the integration with the third party channels like Telegram. There are some aspects which could be improved in the future as well, for example we can enlarge and enrich the functions of the API or connect our chatbot with more APIs; and we can modify and make improvements to our rasa_nlu and rasa_core training data to make our bot more intelligent and anthropopathic.