Weather Chatbot

Guo Ziyu

2019-9-22

Contents

Introduction

This project aims in building a chatbot on WeChat (a popular social app) to provide convenient weather service including current weather report, forecast of future 5 days by intervals of 3 hours and short advice based on the forecast. Building the model and implementing it on WeChat requires using intent and entity recognition. This can be done by importing RasaNLU framework. This project is followed by Dr. Fan Zhang at MIT IBM Watson AI Lab.

Procedures

Backgrounds

In recent years, great progress has been made in image recognition using deep learning techniques, which allows many applications involving image processing to appear. However, natural language processing (NLP) is facing more difficulties comparing with image processing. In the project I learned basic NLP concepts including intent recognition, entity recognition and application of deep learning in NLP. To integrate what I have learned, I built this chatbot which provide basic weather forecast service on WeChat.

Detailed Steps

Data and Goal

We collect weather data from OpenWeatherMap API. Code below shows how to get current weather data and forecast of future 5 days.

```
headers = {
    'x-rapidapi-host': "community-open-weather-map.p.rapidapi.com",
    'x-rapidapi-key': "0082dc76cdmsh64dccca57a89563p15b29ejsnea4004879ee4"
    }
```

```
url = "https://community-open-weather-map.p.rapidapi.com/weather"
querystring = {"callback": "test", "id": "2172797", "units": "\"metric\" or \"imperial\"", "mode": "xml, html",
             "q": city}
response = requests.request("GET", url, headers=headers, params=querystring)
```

```
url = "https://community-open-weather-map.p.rapidapi.com/forecast"
querystring = {"q": city}
response = requests.request("GET", url, headers=headers, params=querystring)
```

After we get data from OpenWeatherMap, we need to identify its content and transform it into json format for future using. An example response from OpenWeatherMap is shown

below.

test({"coord":{"lon":-0.13,"lat":51.51},"weather":[{"id":520,"main":"Rain","description":"light intensity shower rain","icon":"09d"}],
"base":"stations","main""sys":{"type":1,"id":1414,"message":0.0111,"country":"GB","sunrise":1569131156,"sunset":1569175255},
"timezone":3600,"id":2643743,"name":"London","cod":200})

Another data set we need is collection of sentences that people usually use when asking about weather. These sentences need to be labeled with intents and entities in them. I personally made those sentences and used them as training data for training language model. Part of the collection is shown below.

```
{
  "text": "great choice",
  "intent": "affirm",
  "entities": []
},
{
  "text": "sounds really good",
  "intent": "affirm",
  "entities": []
},

{
  "text": "I live in London.",
  "intent": "choose_city",
  "entities": [
    {
      "start": 10,
      "end": 16,
      "value": "London",
      "entity": "city"
    }
  ]
},
```

In this project, we need to identify Four main intents: choosing users' cities, asking for current weather data, asking for forecast and asking for advices. Also, we need our model to identify cities that users choose.

RasaNLU Framework

Rasa NLU is an open-source natural language processing tool for intent classification and entity extraction. To identify entities and intents, we first need to feed the model with labeled data.

```
# trainer = Trainer(config.load("config_spacy.yml"))

# # Load the training data
# training_data = load_data('trainingdata.json')

# # Create an interpreter by training the model
# interpreter = trainer.train(training_data)
```

This is a tricky part because you need enough data to teach the model. In my case, I didn't have enough time to generate much data, and it turns out that it cannot identify certain sentences. For example, if in the collection there are sentences like 'I am in Paris now' and 'Tell me the weather of Shanghai.', both are labeled with intent 'choose_city', and the model will identify sentences expressing that the user lives in Paris or they want to know the weather of Shanghai as 'choose_city'. But if the user want to know the weather of Paris but there are

no sentences like 'How about the weather of Paris', it will not be able to tell its intent. This can be easily be solved by adding more training data, which will be done in future. Code for implementing rasa model for interpreting sentences are shown below.

```
def interpret_msg(message):
    msg=interpreter.parse(message)
    intent=msg['intent']['name']
    if len(msg['entities'])!= 0:
        entity=msg['entities'][0]['value']
    else :entity = "none"
    return intent, entity
```

Response Generating

After the model is trained and able to identify information we need, we shall generate answer to uses' messages based on weather data got from OpenWeatherAPI. The Chatbot has 2 state: INIT and CHOOSED. In the beginning, the bot is at INIT state, which askes users to choose the city they what to search. After the city is chosen, the bot enters CHOOSED state, and provide current weather information of the city as well as wait for the user to give further instructions. Then, the bot will identify uses' intent and give corresponding answers including forecast, detailed information and choosing another city.

give_advice: give advices based on raining and temperature forecast

```
def give_advice(message):
    result = ""
    rain = 1
    temp = 0
    for i in range(8):
        if message['list'][i]['weather'][0]['main'] == 'Rain':
            rain = rain*0
        else : rain = rain*1
        temp = temp + message['list'][i]['main']['temp']-273
    temp = temp/8
    if rain == 1 :
        result= result + "There might be rains in the next 24 hours, please take your umbrella."
    else:
        result = result+ "The possibility of raining in the next 24 hours is low."
    if temp < 0:
        result = result + " Tommorrow will be very cold, please keep warm."
    if temp<10:
        result = result + " Tommorrow will be cold, be careful of getting cold."
    elif temp<18:
        result = result + " Tommorrow will be a cool and nice day."
    elif temp< 26:
        result = result + " Tommorrow wil be a warm and nice day."
    elif temp<32:
        result = result + " Tommorrow will be hot, please drink enough water."
    else :
        result = result + " Tommorrow will be very hot, be careful of getting heat stroke."
    return result
```

response_gen: generate response based on information collected from API and users' intent

```
def response_gen(city, intent):
    global choosed_response
    global choosed_city
    if intent == "choose_city":
        choosed_city = city
        url = "https://community-open-weather-map.p.rapidapi.com/weather"
        querystring = {"callback": "test", "id": "2172797", "units": "\"metric\" or \"imperial\"", "mode": "xml, html",
                       "q": city}
        response = requests.request("GET", url, headers=headers, params=querystring)
        response = response.text
        choosed_response = response
        response = json.loads(response[5:len(response) - 1])
        result = "The current weather of " + city + " is " + response['weather'][0][
            'description'] + ". the average temperature is " + str(
            response['main']['temp']) + "K, the highest temprature is " + str(
            response['main']['temp_max']) + "K, the lowest temprature is " + str(response['main'][
                                                                                     'temp_min']) + "K. Would you like to know more detailed weather or forecast of next 5 days?"

    elif intent == "detailed":
        response = json.loads(choosed_response[5:len(choosed_response) - 1])
        result = "The humidity is " + str(response['main']['humidity']) + ", the pressure is " + str(
            response['main']['pressure']) + ", the visibility is " + str(
            response['visibility']) + ", the wind speed is " + str(
            response['wind']['speed']) + ", the wind direction is degree " + str(
            response['wind']['deg']) + ". Is there anything else you what to know?"

    elif intent == "forecast":
        url = "https://community-open-weather-map.p.rapidapi.com/forecast"
        querystring = {"q": city}
        response = requests.request("GET", url, headers=headers, params=querystring)
        city = choosed_city
        response = json.loads(response.text)
        result = ""
        for i in range(40):
            result = result + response['list'][i]['dt_txt'] + "\n"
            result = result + response['list'][i]['weather'][0]['description'] + " " + str(
                response['list'][i]['main']['temp']) + "K\n"
    elif intent == "advice":
        url = "https://community-open-weather-map.p.rapidapi.com/forecast"
        city = choosed_city
        querystring = {"q": city}
        response = requests.request("GET", url, headers=headers, params=querystring)
        response = json.loads(response.text)
        result = give_advice(response)
    return result
```

interpret_msg: identify intent and entities in uses' messages

```
def interpret_msg(message):
    msg=interpreter.parse(message)
    intent=msg['intent']['name']
    if len(msg['entities'])!= 0:
        entity=msg['entities'][0]['value']
    else :entity = "none"
    return intent, entity
```

policyrule: define state transformation and responses generation rules

```
def policyrule():
    policy = {
        (INIT, "none"): (INIT, "I'm sorry - I'm not sure how to help you."),
        (INIT, "choose_city"): (CHOOSED, "require_generating"),

        (CHOOSED, "none"): (CHOOSED, "I'm sorry - I'm not sure how to help you."),
        (CHOOSED, "choose_city"): (CHOOSED, "require_generating"),
        (CHOOSED, "advice"): (CHOOSED, "require_generating"),
        (CHOOSED, "detailed"): (CHOOSED, "require_generating"),
        (CHOOSED, "forecast"): (CHOOSED, "require_generating"),
        (CHOOSED, "thankyou"): (INIT, "you are welcome."),
    }
    return policy
```

respond and send_message : integrate functions above and give responses to users

```
def respond(state, message, policy):
    intent, entity = interpret_msg(message)
    (new_state, response) = policy[(state, intent)]
    if response == "require_generating":
        response = response_gen(entity, intent)
    return new_state, response
```

```
def send_message(message):
    global state
    state,msg = respond(state, message, policyrule())
    return msg
```

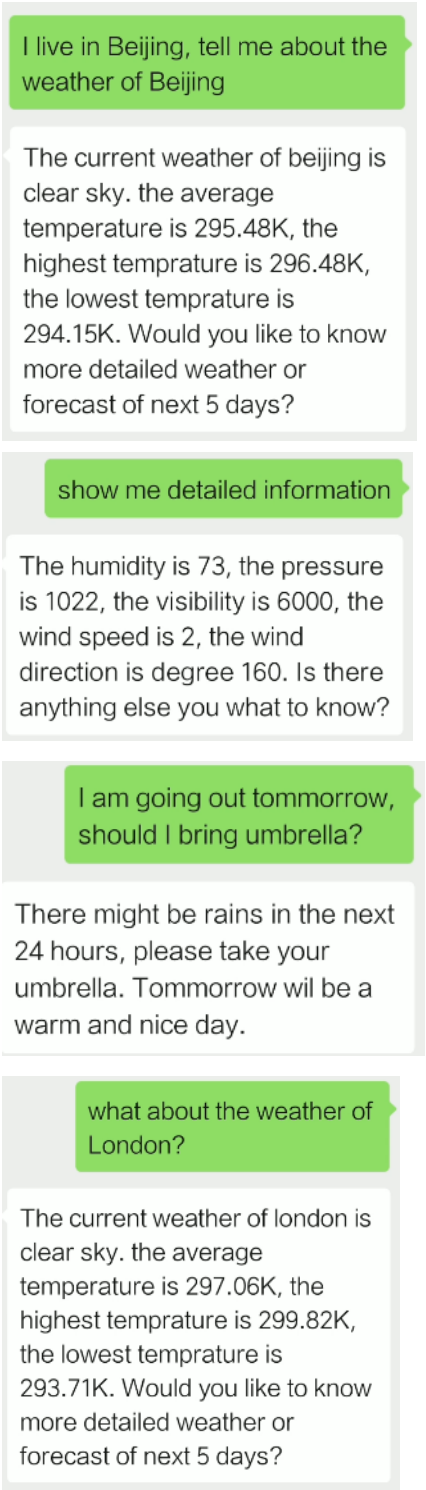
Constructing Chatbot on WeChat

We import wxpy to construct our chatbot. Corresponding code is shown below. This part of code can be replaced to transfer the bot to other apps like Telegrams.

```
from wxpy import *

bot = Bot()

my_friend = bot.friends().search('Guo_')[0]


@bot.register(my_friend)
def my_friednd_message(msg):
    msg = str(msg.text)
    return send_message(msg)


embed()
```

Result

I tested the bot with a few questions and the bot worked well. To improve the bot's performance on more questions, I need to get more data of sentences people use when refer to the weather. More results can be found in Chatbot.mp4 in the folder.



Conclusion

Even that I built the bot myself, I am amazed by the feeling that I am doing chat with an entity when searching for information, not staring at tables of numbers. However, this bot can only answer sentences that are similar to sentences that they are trained before, which limits its application in more complex scenarios. Also, the data they need requires much efforts for

labeling, which can be very difficult when building more ‘clever’ bots. This means we need better self-learning model including Reinforcement Learning to improve its performance in the future.