# Natural Language Analysis Chatbot Project

## ——A Weather Query Chat Bot Based On Telegram Bot

JiaWei Qiu

ShanghaiTech University

Electronic Engineering

qiujw@shanghaitech.edu.cn

# Contents

**Abstract**

This report aims to briefly analyze the working principle of natural language processing chatbot and introduce an implementation of a chatbot helping people to search weather information.

**keywords:** artificial intelligence, natural language analysis, Rasa-NLU, machine learning, chatbot, weather

# 1 Background

With the continuous development of science and technology, artificial intelligence has begun to play an increasingly important role in people's life.Many companies have been developing intelligent chatbots, such as WeChat xiaobing, QQ xiaobing, siri...Not only can they chat with us autonomously, sometimes they can even ask us some important questions, which can be said to be very human and intelligent products.This has also generated widespread interest in the market.Intelligent chatbots exist in many fields, such as weather forecast, stock query, football score prediction and so on.

There are many ways to realize intelligent robot. I used python language to program these projects, and the following methods were applied to complete the normal dialogue of robot.

1) multiple alternative answers to the same question, and the scheme of default answers is provided;

2) can answer questions through regular expressions, pattern matching, keyword extraction, syntactic conversion, etc.

3) can extract user intention through one of regular expression, nearest neighbor classification and support vector machines or more;

4) identify named entities through pre-built named entity types, role relationships, dependency analysis...

5) construction of local basic chatbot system based on Rasa NLU;

6) database query and use natural language to explore database contents (extract parameters, create queries, and respond)

7) single-round multiple incremental query technology based on incremental filter and entity discrimination denial technology

8) realize the multi-round and multi-query technology of the state machine, and provide explanations and answers based on contextual questions

9) multi-round multi-query techniques for handling rejections, waiting state transitions, and pending actions

In addition, I integrated the robot into telegram.Telegram can set up a personal robot by itself. It is very convenient and simple to compile telegram only by acquiring its token and calling it.

For the implementation of weather forecast, the API we use is: (1) https://rapidapi.com/community/api/open-weather-map?endpoint=53aa6043e4b00287471a2b66 and (2) https://rapidapi.com/community/api/open-weather-map?endpoint=53aa6041e4b00287471a2b62. The former can provide us with weather forecast in any location on the earth. The flexible algorithm of weather calculation let it provide weather data not only for cities but for any geographic coordinates. It is important for megapolices, for example, where weather is different on opposit city edges. We can get forecast data every 3 hours or daily. The 3 hours forecast is available for 5 days. All weather data can be obtained in JSON or XML format. Then the later one provides us with current weather data still in any location on the earth. The current weather data are updated online based on data from more than 40,000 weather stations. It's very powerful.

We use pyTelegramBotAPI package to compile the robot.

# 2 Principle

The main principles we used this time are: several rounds of talks, regular expressions, nearest neighbor classification, support vector machines, named entity recognition, incremental filter.

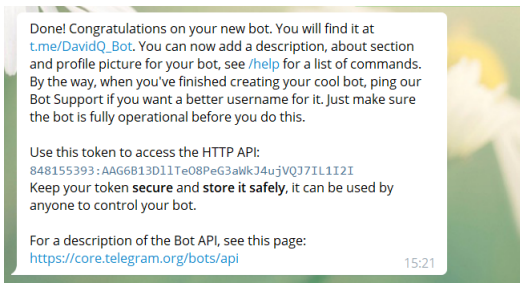
python, spacy, Rasa NLU, pyTelegramBotAPI

# 3 Code Analysis

## 3.1 Start your bot

First of all, I would like to introduce how to set up a robot of my own through telegram.

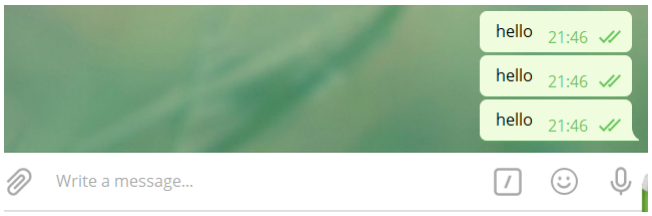
We need to search for Bot_Father in the search box. Then follow the steps to create your own bot. BotFather will return you a Token:




```
27  TOKEN = '[redacted]'
28
29  bot = telebot.TeleBot(TOKEN)
```

start a bot

At this point we can talk to the robot, but it won't respond.




chat with the bot

So we need to program so that the robot can respond us.

Here are the custom functions and python packages I call. I'm using telebot package of pyTelegramBotAPI to realize the robot.

```
1  # -*- coding: utf-8 -*-
2  # Import necessary modules
3  import telebot
4  from telebot import types
5  import re
6  import random
7  import datetime
8  import requests
9
10
11 from rasa_nlu.training_data import load_data
12 from rasa_nlu.config import RasaNLUModelConfig
13 from rasa_nlu.model import Trainer
14 from rasa_nlu import config
15
```

import packages

## 3.2 Set /start and /help

First we'll use my own token and call it. Following my experience chatting with Bot_Father, I first set up two shortcuts command for my robot: /start and /help.Enter these two shortcut command in the dialog and the robot will reply with a paragraph.When we type /start, it tells us about its profile, and it prompts us to type /help for more information.When we type /help, it will tell us some of its main functions.

We can see that there is a decorator above the function. By using this decorator, we can call the message_handler function in the telebot package.Command = ['help'] for setting this function is /help. Using

```
#get start
@bot.message_handler(commands=['start'])
def bot_start(message):
    if message.text == '/start':
        bot.send_chat_action(message.chat.id, 'typing')
        bot.send_message(message.chat.id, 'Hello, I am bot DavidQ. :) \n'
                                          'I can chat with you as a friend. You can say whatever you want to me!  (￣▽￣) \n'
                                          'More importantly! I can give you some information about the weather of the cities you want to know
                                          'If you what more information about my command, just tell me /help. (￣ε(￣)' , reply_markup=replyKe
    else:
        bot.send_chat_action(message.chat.id, 'typing')
        bot.send_message(message.chat.id, 'Sorry my dear friend. I am not sure if you what to start chatting with me. :(\n'
                                          'Please enter /start to chat with me! ', reply_markup=replyKeyboard.startKeyboard())
```

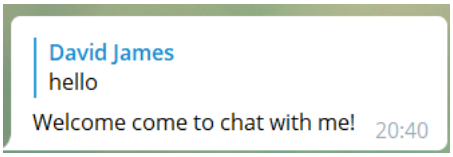
set /start

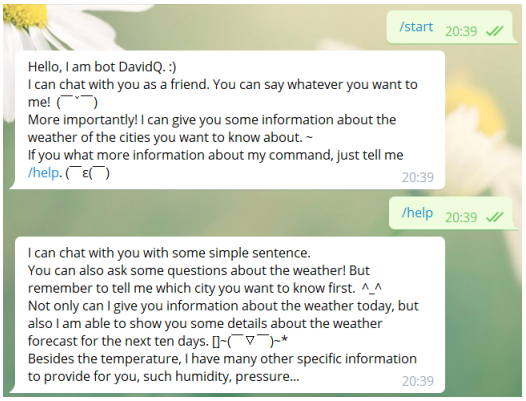
```
#want get some help
@bot.message_handler(commands=['help'])
def bot_help(message):
    if message.text == '/help':
        bot.send_chat_action(message.chat.id, 'typing')
        bot.send_message(message.chat.id, 'I can chat with you with some simple sentence.\n'
                                          'You can also ask some questions about the weather! But remember to tell me which city you want to know first.  ^_^ \n'
                                          'Not only can I give you information about the weather today, but also I am able to show you some details about the weather forecast for the next ten
                                          'Besides the temperature, I have many other specific information to provide for you, such humidity, pressure... \n',
                                          'If you doesn\'t tell me city or date, I will set city to Shanghai and date to today as defaults. \n'
                                          'Am I powerful? <(￣﹀￣)>  Can\'t wait to start talking to me? Let\'s get it!',reply_markup=replyKeyboard.startKeyboard())
    else:
        bot.send_chat_action(message.chat.id, 'typing')
        bot.send_message(message.chat.id,
                         'Sorry my dear friend. I am not sure if you what to get some help and know more details about the functions of mine. ?_?\n'
                         'Please enter /help to get help! ^_^', reply_markup=replyKeyboard.helpKeyboard())
```

set /help

send_chat_action, we can set the action to be typing as the robot prepares to respond, just as we do when someone responds to our messages on WeChat. We use send_message to reply to messages. We only need to provide message.chat.id and the reply statement to command the robot to reply.

In addition, the telebot has a reply_to() function to reply messages, which requires us to provide the message received and the statement to reply, so the effect will be more like a real conversation, so I used this function to command the robot to reply in subsequent conversations.




reply and input /start ors /help

So we've done the first step.

Next I set some keyboard parameters.

## 3.3 Build function to chat

Then, I'm going to write a real conversation function.

The echo () function handles all incoming messages.When the robot receives a message, it will return us a long dictionary containing the account number, user name of the person who sent the message, message content and so on , rather than simply returning the received message.So we need to use the message.text statement

```python
class replyKeyboard():
    def startKeyboard():
        markup_start = types.ReplyKeyboardMarkup(row_width=2)
        itembtn1 = types.KeyboardButton('/start')
        markup_start.add(itembtn1)
    def helpKeyboard():
        markup_help = types.ReplyKeyboardMarkup(row_width=2)
        itembtn2 = types.KeyboardButton('/help')
        markup_help.add(itembtn2)
```

set keyboard

to convert the contents received by the function into a string, so that we can get the message received by the robot for later operation.

```python
@bot.message_handler()
def echo(message):
    #print(message)#print(type(message))
    message_text = message.text #print(message_text)
    # Check if the message is in the ask_name
```

get message

```
{'content_type': 'text', 'message_id': 109, 'from_user': {'
<class 'telebot.types.Message'>
```

received message and type

We can find that the type of message function received first is not a string but a class 'telebot.types.Message'. So it is necessary to change it to text.

### 3.3.1 Ask name

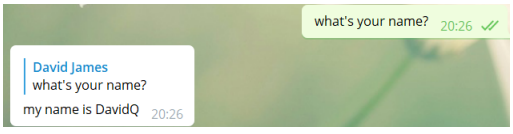
After that, we have to judge the message.First, when the message is asking for the robot's personal information,by determining if the message is exactly the same as the key in the ask_name dictionary. If the answer is true, the function will select a random answer from the ask_name list and reply it.

```python
    if message_text in ask_name:
        # Return a random matching response
        bot_message = random.choice(ask_name[message_text])
        bot.reply_to(message, bot_message)
        return None
```

a function to reply when asking for robot's name

```python
name = "DavidQ"

# Define a dictionary containing a list of ask_name for each message
ask_name = {
  "what's your name?": [
      "my name is {0}".format(name),
      "they call me {0}".format(name),
      "I go by {0}".format(name),
      "you can call me {0}".format(name)
  ],
  "Can you tell me your name?": [
      "my name is {0}".format(name),
      "they call me {0}".format(name),
      "I go by {0}".format(name),
      "you can call me {0}".format(name)
  ],
}
```

ask_name




ask_name result

In order to prevent subsequent programs from processing the message to reply unsatisfactory messages, when the reply is sent by bot successfully and correctly, the function will return None to end this whole function.

### 3.3.2 Match Rules and Respond

Next, we'll help the robot respond to a few simple everyday conversations.First we'll set up a dictionary, where keys begin a sentence and values are the responses.

```
rules = {'I want (.*)': ['What would it mean if you got {0}',
                         'Why do you want {0}',
                         "What's stopping you from getting {0}"],
         'do you remember (.*)': ['Did you think I would forget {0}',
                                  "Why haven't you been able to forget {0}",
                                  'What about {0}',
                                  'Yes .. and?'],
         'do you think (.*)': ['if {0}? Absolutely.',
                               'No chance',
                               'of course! I always think {0}'],
         'if (.*)': ["Do you really think it's likely that {0}",
                     'Do you wish that {0}',
                     'What do you think about {0}',
                     'Really--if {0}']
         }
```

rules

```
reply = respond_rules(message_text)
if reply != message_text:
    bot.reply_to(message, reply)
    return None
```

a function to reply the rules

We use respond_rules and match_rules.

```
def respond_rules(message):
    # Call match_rule
    response, phrase = match_rule(rules, message)
    if response != 'default':
        if '{0}' in response:
            # Replace the pronouns in the phrase
            phrase = replace_pronouns(phrase)
            # Include the phrase in the response
            response = response.format(phrase)
        return response
    else:
        return message
```

a function to reply the rules

```
def match_rule(rules, message):
    for pattern, responses in rules.items():
        match = re.search(pattern, message)
        if match is not None:
            response = random.choice(responses)
            var = match.group(1) if '{0}' in response else None
            return response, var
    return 'default', None
```

match_rules

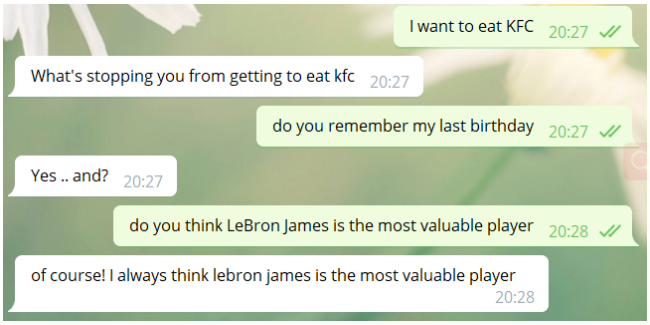
Match_rules matches the received message with the keys of dict(rules) one by one through a regular expression. If the match is successful, we randomly select the corresponding answer as response.At the same time, the symbol '(.*)' means greedy matching. The regular expression will match the first part of the sentence as group(0), and the second part of the sentence as group (1).When we need to call it, we just do it like that photo.When there is a 0 in the response, this represents that we need to have content formatted to 0. Then we need to use the group(1) and call replace_pronouns funnction, which will change I to you , you to me, and so on.

```
def replace_pronouns(message):
    message = message.lower()
    if 'me ' in message:
        return re.sub('me', 'you', message)
    if 'i ' in message:
        return re.sub('i', 'you', message)
    elif 'my ' in message:
        return re.sub('my', 'your', message)
    elif 'your ' in message:
        return re.sub('your', 'my', message)
    elif 'you ' in message:
        return re.sub('you', 'me', message)

    return message
```

replace_pronouns

If we simply find me in the sentence and replace it, there will be mistakes like changing James to Jayous. So I add some space in front of or behind these words. This still doesn't avoid all the mistakes, but it's a lot better than before.

When the regular expression match fails initially, the function returns' default 'and none.The next few functions also do not operate on message.




match_rules results

### 3.3.3 Rasa Train and Entity Identification

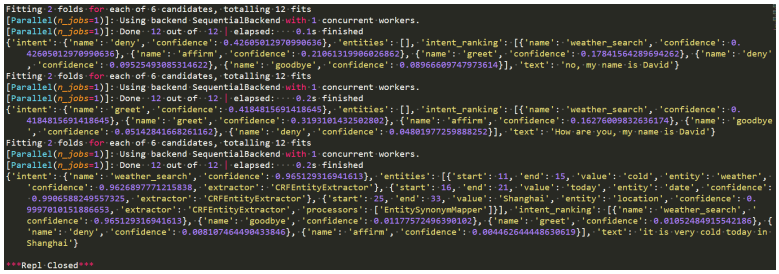
When neither dict matches successfully, we come to the third step.

First we will train message by rasa. I wrote a rasa training demo myself, which set some synonyms and several major intent, such as greet, weather_search,goodbye, deny and affirm. There are also many entities, such as location, weather and date. Location records the city name in the statement.Date records the message asks the weather on which day.Weather includes rain, hot, cold, mild, cloudy and so on. Through rasa training, we can basically get accurate entity recognition and intention recognition for statements.

```
160    def rasa_train(message):
161        training_data = load_data('demo-rasa.json')
162        # Create a trainer
163        trainer = Trainer(config.load("config_spacy.yml"))
164        # Create an interpreter by training the model
165        interpreter = trainer.train(training_data)
166        response = interpreter.parse(message)
167        matched_intent = None
168        for intent, pattern in patterns.items():
169            if re.search(pattern,message) is not None:
170                matched_intent = intent
171                response["intent"]["name"] = matched_intent
172        return response
```

rasa_train

However, because we can get the exact weather_search intention in most cases, but the greet, goodbye, etc., cannot be easily identified, I set up a dict(patterns) to match the obvious non-weather_search intention in the sentence.We use regular expressions to match, and if the match succeeds, we override the intent to the correct intent.




train result

From this result we can clearly find the intention of the message.

And then I'm going to go through how to respond each intent one one by one.

### 3.3.4 Greet and Goodbye

```
477         data = rasa_train(message_text)
478         #print(data)
479         if data['intent']['name'] == 'greet':
480             # Find the name
481             name = find_name(message_text)
482             if name is None:
483                 answer = random.choice(say_hello["no_name"])
484             else:
485                 answer = random.choice(say_hello["with_name"]).format(name)
486             bot.reply_to(message, answer)
487             return None
```

when the intent is greet

When the intent is greet, we'll use the find_name function to find out if there's a name in the greeting message. We use regular expressions to find out if there are two keywords in the statement: name and call.If true, we continue to use regular expressions to find all word or words in a sentence that begin with an uppercase letter, followed by any number of lowercase letters, because that's the format for most people's names.I then compose all the names that matched into a string by '.join ().But in order to avoid matching the words Hello, Hi, and so on at the beginning of the sentence, I will match the name that matched before with the words at the beginning of the sentence.If true, replace all of these words in name by re. sub with empty.

```
174 def find_name(message):
175     name = None
176     # Create a pattern for checking if the keywords(name and call) occur
177     name_keyword = re.compile('name|call')
178     # Create a pattern for finding capitalized words
179     name_pattern = re.compile('[A-Z]{1}[a-z]+')
180     if name_keyword.search(message):
181         # Get the matching words in the string
182         name_words = name_pattern.findall(message)
183         if len(name_words) > 0:
184             # Return the name if the keywords are present
185             name = ' '.join(name_words)  #many words in a list change to a str
186             sent_head = "Hello|Hi|Hey|What's|How|Why|What|Who|My"
187             if re.search(sent_head,name) is not None:
188                 delete = re.findall(sent_head,name).pop()+' '
189                 name = re.sub(delete,'',name)
190     return name
```

find_name

If a name is found, reply a message with that person's name, or reply a message with no name.

```
110 say_hello = {
111     "with_name": ["What's up! {0}",
112                   "Welcome to the new world! {0}",
113                   "I am happy to chat with you, dear {0}",
114                   "Hello, {0}!",
115                   "Hi, {0}! Please enjoy yourself here!"],
116     "no_name":["I am interested in being your friend",
117                "Welcome come to chat with me!",
118                "Hello my friend! My name is DavidQ",
119                "Nice to meet you!",
120                "How are you!"]
121 }
```

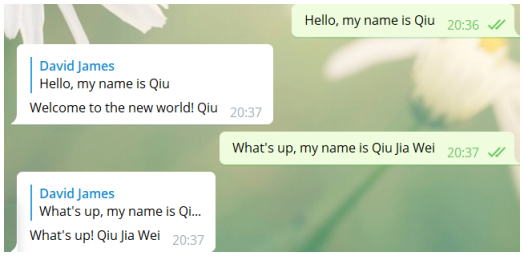
say_hello

Similarly, when the intent is goodbye, the robot replies to the random statement in say_goodbye.

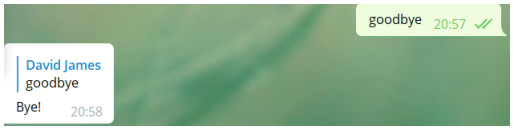
```
say_goodbye = {'bye':["Alright, byebye!",
                      "Goodbye my friend!",
                      "See you next time!",
                      "Bye!"]}
```

```
if data['intent']['name'] == 'goodbye':
    answer = random.choice(say_goodbye['bye'])
    bot.reply_to(message, answer)
    return None
```

goodbye




greet result




goodbye result

### 3.3.5 Weather Search

With none of these judgments true, we are left with the last intention, weather_search.First, we extract entities from rasa training results and put them into params, a dict.This is a global variable.So we can modify the dict over and over again over several rounds of conversations until it meets the final requirements.

```
if data['intent']['name'] == 'weather_search':
    entities = data['entities']
    for ent in entities:
        params[ent["entity"]] = str(ent["value"])
    if "location" not in params and "date" in params:
        interpret = "no_location"
        state, respond = policy[(state,interpret)]
        bot.reply_to(message, respond)
    elif "location" in params and "date" not in params:
        interpret = "no_date"
        state, respond = policy[(state,interpret)]
        bot.reply_to(message,respond)
    elif "location" not in params and "date" not in params:
        interpret = "no_location_date"
        state, respond = policy[(state, interpret)]
        bot.reply_to(message, respond)
    elif "location" in params and "date" in params:
        interpret = "get_location_date"
        state, respond = policy[(state, interpret)]
        bot.reply_to(message, respond)
    if state == FIND_WEATHER:
        location = params["location"]
        date = params['date']
        weather = None
        if 'weather' in params:
            weather = params["weather"]
```

```
        state, respond = policy[(state, interpret)]
        bot.reply_to(message, respond)
    if state == FIND_WEATHER:
        location = params["location"]
        date = params['date']
        weather = None
        if 'weather' in params:
            weather = params["weather"]
        if date == "today":
            current_weather(message, location, weather)
        elif date == "tomorrow":
            forecast_weather(message, location, 1, weather)
        elif "two" in date:
            forecast_weather(message, location, 2, weather)
        elif "three" in date:
            forecast_weather(message, location, 3, weather)
        elif "four" in date:
            forecast_weather(message, location, 4, weather)
        elif "five" in date:
            forecast_weather(message, location, 5, weather)
        state = INIT
        params ={}
    print(state)
    return None
```

when the intent is weather_search

Since the robot must know both the city and the date when it is looking for the weather, we must ensure that the weather query is not executed until we have both location and date keys in params. Otherwise, the robot will return a prompt for the user to provide more messages.

Let me show you how to implement multiple rounds of dialogue.

First, we set the following states and policy.

```
INIT = 0
WANT_LOCATION = 1
WANT_DATE = 2
WANT_LOCATION_DATE =
FIND_WEATHER = 4
global params
params = {}
global state
state = INIT
```

state

```
policy = {
    (INIT, "no_location"): (WANT_LOCATION, "sorry, you didn\'t tell me which city did you want to know about! \n so please tell me more informa
    (INIT, "get_location_date"): (FIND_WEATHER, "Well, please wait for a few seconds!"),
    (INIT, "no_date"):(WANT_DATE, "sorry, it seems that you didn't tell me which day's weather do you want to know about"),
    (INIT, "no_location_date"): (WANT_LOCATION_DATE, "Excuse me, I am not sure which city and which day's weather do you want to know!"),
    (WANT_LOCATION, "get_location_date"): (FIND_WEATHER, "perfect, Please wait a few seconds!"),
    (WANT_LOCATION, "no_location"): (WANT_LOCATION, "sorry I still don't know the city you want to know about!"),
    (WANT_LOCATION,"no_date"): (WANT_DATE,"sorry, though I know the location you want to find, you still need to tell me which day's weather do
    (WANT_DATE,"get_location_date"): (FIND_WEATHER, "excellent! please wait for a few seconds!"),
    (WANT_DATE, "no_date"): (WANT_DATE, "sorry I still don't know the date you want to know about!"),
    (WANT_LOCATION_DATE,"get_location_date"):(FIND_WEATHER,"Fantastic, please wait a few seconds!"),
    (WANT_LOCATION_DATE,"no_location_date"):(WANT_LOCATION_DATE,"sorry, I still do not know either the city or date you want!")
}
```

policy

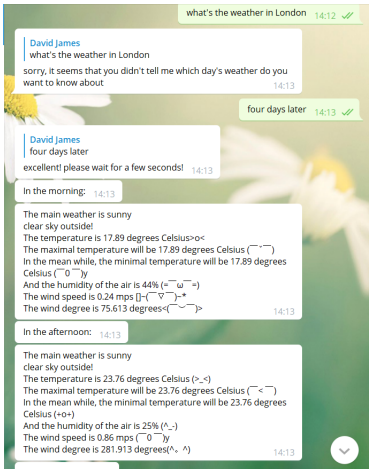
Each time the robot receives a message, it analyzes whether it has both location and date.What is missing prompts the user to provide the missing information.For example, missing only location without missing date, the program tells the robot now interpret as no_location, and the robot finds the new state (WANT_LOCATION) and respond to the user in policy based on what is now state (INIT) and this interpret, and replies to this respond.Because state is also a global variable, it will maintain the previous value every time the function is called, and will not be reset.

When the user knows that he needs to provide more information, he will send corresponding messages to the robot.At this point, the robot will entity recognize the message again.In this example, interpret becomes get_location_date if the user's second message provides location, then both location and date are in params.The new state is also FIND_WEATHER, which means the robot can find weather information.

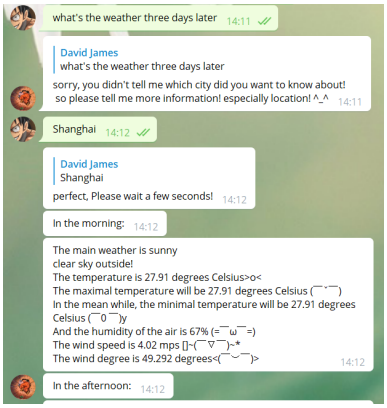
Finally, the program calls either current_weather or forecast_weather, depending on whether the date is today or tomorrow or even farther in the future (up to five days).After a successful round of weather queries, the program manually resets state and params to their original values, INIT and .

It is worth mentioning that since state and params are global variables, if the robot suddenly accepts a sentence that has nothing to do with location or date in multiple rounds of conversations, it will not affect the process of the whole conversation. As long as the user provides location or date afterwards, no matter how many irrelevant sentences there are, there will be no problem.

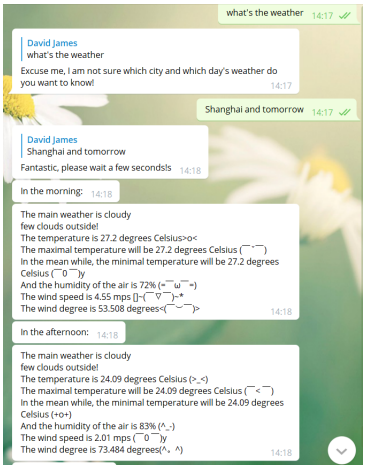
Here are some results.



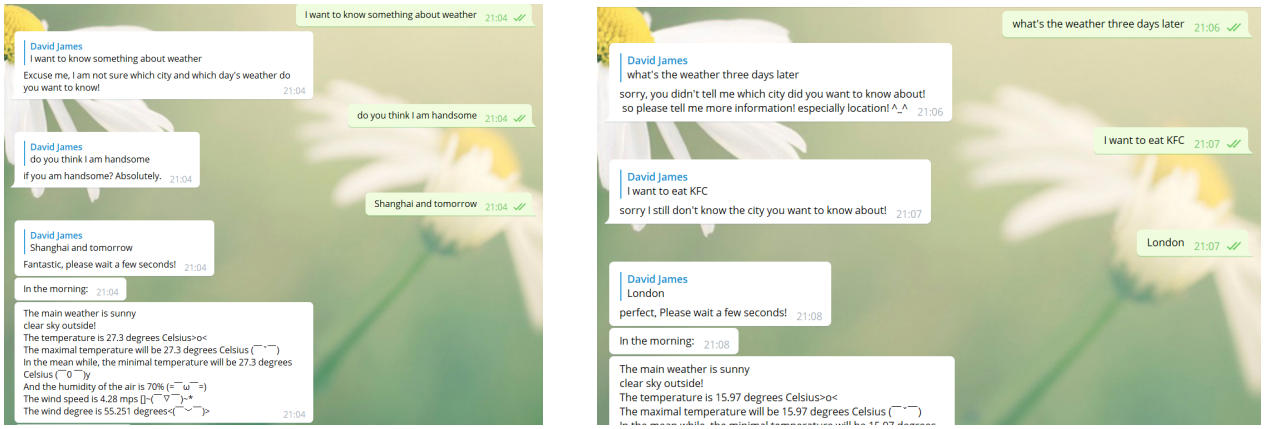

Multiple rounds of conversations with no date




Multiple rounds of conversations with no location




Multiple rounds of conversations with no location and date




Multiple rounds of conversations with unrelated messages

Let's also take a look at the current_weather and forecast_weather functions.

```python
def current_weather(message,city = "Shanghai", weather = None):
    global real_weather
    url = "https://community-open-weather-map.p.rapidapi.com/weather"
    querystring = {"callback": "test", "id": "2172797", "units": "\"metric\" or \"imperial\"", "mode": "xml, html","q": "Shanghai"}
    try:
        querystring['q'] = str(city)

        headers = {
            'x-rapidapi-host': "community-open-weather-map.p.rapidapi.com",
            'x-rapidapi-key': "925d2685d3msha9d08e5660f54e7p129f08jsn9d1ee9a65b48"
            }

        response = requests.request("GET", url, headers=headers, params=querystring)
        response_text = response.text
        #print(response_text)
        r_t_del1 = re.sub('test'+'\(','',response_text)
        r_t_del2 = re.sub('\)','',r_t_del1)
        response_dict = eval(r_t_del2)
        #print(response_dict)

        main_weather = response_dict['weather'][0]['main']
        weather_description = response_dict['weather'][0]['description']
        #print(weather_description)
        temp = round(response_dict['main']['temp'] - 273, 2)
        press = response_dict['main']['pressure']
        humi = response_dict['main']['humidity']
        temp_min = round(response_dict['main']['temp_min'] - 273, 2)
        temp_max = round(response_dict['main']['temp_max'] - 273, 2)
        visi = response_dict['visibility']
        wind_spe = response_dict['wind']['speed']
        #wind_deg = response_dict['wind']['deg']
        #print(temp,press,humi,temp_min,temp_max,visi,wind_spe,wind_deg)

        if main_weather == 'Clouds':
            real_weather = 'cloudy'
        elif main_weather == 'Rain':
            real_weather = 'rainy'
        elif main_weather == 'Clear':
            real_weather = 'sunny'

        reply0 = 'Today, the weather is mainly {0}'.format(real_weather) + '\n'
        reply00 = '{0} outside!'.format(weather_description) + '\n'
        reply1 = 'Now, the temperature is {0} degrees Celsius'.format(temp) + ' (  ˙  )\n'
        reply2 = 'The maximal temperature will be {0} degrees Celsius'.format(temp_max) + ' <(￣ᵕ￣)>\n'
        reply3 = 'In the mean while, the minimal temperature will be {0} degrees Celsius'.format(temp_min) + ' []~(￣▽￣)~*\n'
        reply4 = 'What\'s more,the pressure of the air is {0} Pa'.format(press) + ' (￣▽￣)~*\n'
        reply5 = 'And the humidity of the air is {0}%'.format(humi) + ' (=￣ω￣=)\n'
        reply6 = 'At the same time, the visibility today is  {0}'.format(visi) + ' (￣3￣)a\n'
        reply7 = 'The wind speed is {0} mps'.format(wind_spe) + ' (￣0 ￣)y\n'
        if temp >= 30:
            reply9 = 'Today is so hot! ~_~ It is a good idea to eat some ice-cream!' + '\n'
        elif temp >= 10 and temp < 30:
            reply9 = 'The temperature today is very mild! ^_^ You can go for a walk or go out for fun!' + '\n'
        else:
            reply9 = 'Today is so cold! T_T Remember to wear enough clothes!' + '\n'

        if wind_spe <= 5:
            reply10 = 'There\'s little wind outside! The wind is very mild!' + '\n'
        elif wind_spe > 5 and wind_spe<= 10:
            reply10 = 'It\'s a little windy outside but not too strong! The wind is very mild!' + '\n'
        else:
            reply10 = 'The wind outside is so strong! Please be careful when you go out' + '\n'
        bot.send_message(message.chat.id,reply0+reply00)
        if weather == 'hot' or weather == 'cold' or weather == 'mid':
            bot.send_message(message.chat.id,reply1+reply2+reply3+reply9)
        elif weather == 'windy':
            bot.send_message(message.chat.id,reply7+reply10)
        elif weather == 'cloudy':
            if real_weather == 'Clouds':
                reply11 = 'It\'s clouy outside.' + '\n'
            else:
                reply11 = 'It\'s not clouy outside.' + '\n'
            bot.send_message(message.chat.id,reply11+reply6)
        elif weather == 'rain':
            if real_weather == 'rainy':
                reply12 = 'It\'s rainy outside. Please remember to bring an umbrella with you when going out!' + '\n'
            else:
                reply12 = 'It\'s not rainy outside! You can just go outside without taking an umbrella.' + '\n'
            bot.send_message(message.chat.id,reply12+reply5)
        else:
            bot.send_message(message.chat.id,reply1+reply2+reply3+reply4+reply5+reply6+reply7)
    except KeyError:
        bot.send_message(message.chat.id,'Sorry, it seems that you did tell me a correct city name. Please try again!  T_T')

```

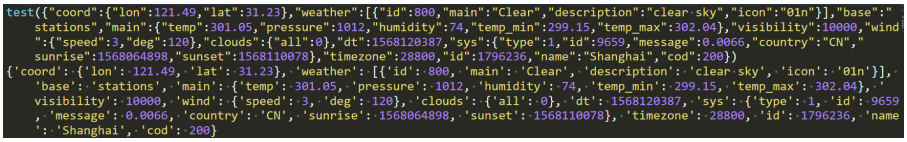
current_weather

```python
def forecast_weather(message,city = "Shanghai", date = 1, weather = None):
    global real_weather_morn, real_weather_noon, real_weather_night
    url = "https://community-open-weather-map.p.rapidapi.com/forecast"

    querystring = {"q": "Shanghai"}
    querystring["q"] = city
    new_day = (datetime.datetime.today()+datetime.timedelta(days=date)).strftime("%Y-%m-%d")

    headers = {
        'x-rapidapi-host': "community-open-weather-map.p.rapidapi.com",
        'x-rapidapi-key': "925d2685d3msha9d08e5660f54e7p129f08jsn9d1ee9a65b48"
        }
    response = requests.request("GET", url, headers=headers, params=querystring)
    response_text = response.text
    r_t_del1 = re.sub('test'+'\(','',response_text)
    r_t_del2 = re.sub('\)','',r_t_del1)
    response_dict = eval(r_t_del2)
    #print(response_dict)

    that_day = []
    morning = '09:00:00'
    afternoon = '15:00:00'
    evening = '18:00:00'
    try:
        for i in range(0,40):
            dt_txt = response_dict['list'][i]['dt_txt']
            a = re.search(new_day+' (.*)',dt_txt)
            if a is not None:
                if a.group(1) == morning or a.group(1) == afternoon or a.group(1) == evening:
                    that_day.append(response_dict['list'][i])


        temp_morn = round(that_day[0]['main']['temp'] - 273, 2)
        temp_min_morn = round(that_day[0]['main']['temp_min'] - 273, 2)
        temp_max_morn = round(that_day[0]['main']['temp_max'] - 273, 2)
        humi_morn = that_day[0]['main']['humidity']
        main_weather_morn = that_day[0]['weather'][0]['main']
        weather_description_morn = that_day[0]['weather'][0]['description']
        wind_spe_morn = that_day[0]['wind']['speed']
        wind_deg_morn = that_day[0]['wind']['deg']

        if main_weather_morn == 'Clouds':
            real_weather_morn = 'cloudy'
        elif main_weather_morn == 'Rain':
            real_weather_morn = 'rainy'
        elif main_weather_morn == 'Clear':
            real_weather_morn = 'sunny'

        bot.send_message(message.chat.id,'In the morning:')
        reply0_m = 'The main weather is {0}'.format(real_weather_morn) + '\n'
        reply00_m = '{0} outside!'.format(weather_description_morn) + '\n'
        reply1_m = 'The temperature is {0} degrees Celsius'.format(temp_morn) + '>o<\n'
        reply2_m = 'The maximal temperature will be {0} degrees Celsius'.format(temp_max_morn) + ' (￣▽￣)\n'
        reply3_m = 'In the mean while, the minimal temperature will be {0} degrees Celsius'.format(temp_min_morn) + ' (￣0 ￣)y\n'
        reply4_m = 'And the humidity of the air is {0}%'.format(humi_morn) + ' (=￣ω￣=)\n'
        reply5_m = 'The wind speed is {0} mps'.format(wind_spe_morn) + ' []~(￣▽￣)~*\n'
        reply6_m = 'The wind degree is {0} degrees'.format(wind_deg_morn) + '<(￣ ￣)> \n'
        bot.send_message(message.chat.id,reply0_m+reply00_m+reply1_m+reply2_m+reply3_m+reply4_m+reply5_m+reply6_m)


        temp_noon = round(that_day[1]['main']['temp'] - 273, 2)
        temp_min_noon = round(that_day[1]['main']['temp_min'] - 273, 2)
        temp_max_noon = round(that_day[1]['main']['temp_max'] - 273, 2)
        humi_noon = that_day[1]['main']['humidity']
        main_weather_noon = that_day[1]['weather'][0]['main']
        weather_description_noon = that_day[1]['weather'][0]['description']
        wind_spe_noon = that_day[1]['wind']['speed']
        wind_deg_noon = that_day[1]['wind']['deg']

        if main_weather_noon == 'Clouds':
            real_weather_noon = 'cloudy'
        elif main_weather_noon == 'Rain':
            real_weather_noon = 'rainy'
        elif main_weather_noon == 'Clear':
            real_weather_noon = 'sunny'

        bot.send_message(message.chat.id,'In the afternoon:')
        reply0_no = 'The main weather is {0}'.format(real_weather_noon) + '\n'
        reply00_no = '{0} outside!'.format(weather_description_noon) + '\n'
        reply1_no = 'The temperature is {0} degrees Celsius'.format(temp_noon) + ' (>_<)\n'
        reply2_no = 'The maximal temperature will be {0} degrees Celsius'.format(temp_max_noon) + ' (￣< ￣)\n'
        reply3_no = 'In the mean while, the minimal temperature will be {0} degrees Celsius'.format(temp_min_noon) + ' (+o+)\n'
        reply4_no = 'And the humidity of the air is {0}%'.format(humi_noon) + ' (^_-)\n'
        reply5_no = 'The wind speed is {0} mps'.format(wind_spe_noon) + ' (￣0 ￣)y\n'
```

```
            reply6_no = 'The wind degree is {0} degrees'.format(wind_deg_noon) + '(^・^)\n'
            bot.send_message(message.chat.id,reply0_no+reply00_no+reply1_no+reply2_no+reply3_no+reply4_no+reply5_no+reply6_no)

            temp_night = round(that_day[2]['main']['temp'] - 273, 2)
            temp_min_night = round(that_day[2]['main']['temp_min'] - 273, 2)
            temp_max_night = round(that_day[2]['main']['temp_max'] - 273, 2)
            humi_night = that_day[2]['main']['humidity']
            main_weather_night = that_day[2]['weather'][0]['main']
            weather_description_night = that_day[2]['weather'][0]['description']
            wind_spe_night = that_day[2]['wind']['speed']
            wind_deg_night = that_day[2]['wind']['deg']

            if main_weather_night == 'Clouds':
                real_weather_night = 'cloudy'
            elif main_weather_night == 'Rain':
                real_weather_night = 'rainy'
            elif main_weather_night == 'Clear':
                real_weather_night = 'sunny'

            bot.send_message(message.chat.id,'In the evening:')
            reply0_ni = 'The main weather is {0}'.format(real_weather_night) + '\n'
            reply00_ni = '{0} outside!'.format(weather_description_night) + '\n'
            reply1_ni = 'The temperature is {0} degrees Celsius'.format(temp_night) + '  ( > v < )\n'
            reply2_ni = 'The maximal temperature will be {0} degrees Celsius'.format(temp_max_night) + ' (O ^ ~ ^ O)\n'
            reply3_ni = 'In the mean while, the minimal temperature will be {0} degrees Celsius'.format(temp_min_night) + ' (>_<)\n'
            reply4_ni = 'And the humidity of the air is {0}%'.format(humi_night) + ' n▽n\n'
            reply5_ni = 'The wind speed is {0} mps'.format(wind_spe_night) + ' Lo(￣▽￣///)\n'
            reply6_ni = 'The wind degree is {0} degrees'.format(wind_deg_night) + ' (～o￣v￣)～o ～\n'
            bot.send_message(message.chat.id,reply0_ni+reply00_ni+reply1_ni+reply2_ni+reply3_ni+reply4_ni+reply5_ni+reply6_ni)

            if temp_noon >= 30:
                reply9 = 'Today is so hot! ~_~ It is a good idea to eat some ice-cream!' + '\n'
            elif temp_noon >= 10 and temp_noon < 30:
                reply9 = 'The temperature today is very mild! ^_^ You can go for a walk or go out for fun!' + '\n'
            else:
                reply9 = 'Today is so cold! T_T Remember to wear enough clothes!' + '\n'

            if wind_spe_noon <= 5:
                reply10 = 'There\'s little wind outside! The wind is very mild!' + '\n'
            elif wind_spe_noon > 5 and wind_spe_noon <= 10:
                reply10 = 'It\'s a little windy outside but not too strong! The wind is very mild!' + '\n'
            else:
                reply10 = 'The wind outside is so strong! Please be careful when you go out' + '\n'


        if weather == 'hot' or weather == 'cold' or weather == 'mid':
            bot.send_message(message.chat.id,reply9)
        elif weather == 'windy':
            bot.send_message(message.chat.id,reply10)
        elif weather == 'cloudy':
            if real_weather_noon == 'Clouds':
                reply11 = 'It\'s clouy outside.' + '\n'
            else:
                reply11 = 'It\'s not clouy outside.' + '\n'
            bot.send_message(message.chat.id,reply11)
        elif weather == 'rain':
            if real_weather_noon == 'rainy':
                reply12 = 'It\'s rainy outside. Please remember to bring an umbrella with you when going out!' + '\n'
            else:
                reply12 = 'It\'s not rainy outside! You can just go outside without taking an umbrella.' + '\n'
            bot.send_message(message.chat.id,reply12)

    except KeyError:
        bot.send_message(message.chat.id,'Sorry, it seems that you did tell me a correct city name. Please try again!  T_T')
    except IndexError:
        bot.send_message(message.chat.id,'Sorry, I can only give you weather forecast in five days. The date you want to know seems to be out of range!  ~_~')
```

forecast_weather

Because the result of our call to the API is string type , we need to modify it with regular expressions and eval () to make it dict.




api result

We use the following statement to find the date of the day, and it is also very easy to find the date of tomorrow.

```
32    today = datetime.date.today().strftime("%Y-%m-%d")
```

```
new_day = (datetime.datetime.today()+datetime.timedelta(days=date)).strftime("%Y-%m-%d")
```

forecast_weather

The try, except statement helps the robot tell the user when it receives an incorrect city name or a date beyond which weather forecasts can be made.

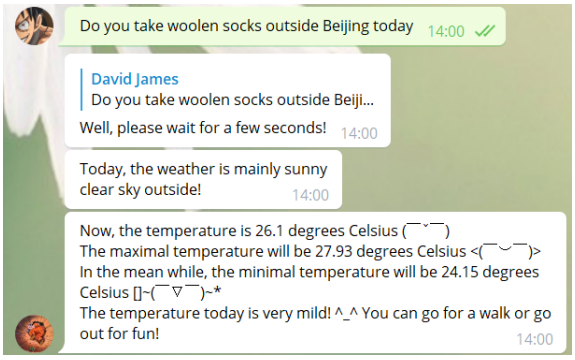
forecast_weather mainly forecasts the weather in the morning, afternoon and evening of the day, which can also help users know more about the weather, rather than just the general weather of the day.

These two functions mainly return the main weather conditions of the day with a description, temperature, maximum temperature, minimum temperature, humidity, custom and visibility.

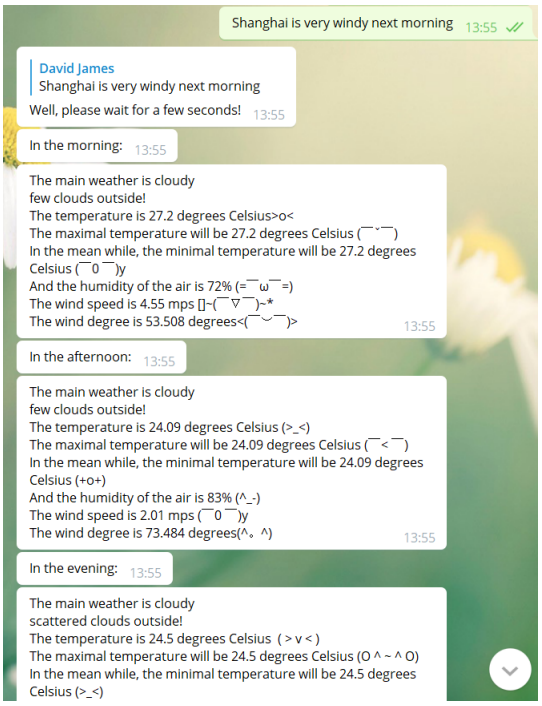
When the user wanted to know the exact weather conditions, such as ”is it hot today?”, the robot also gave more answers.

We all used the reply_to function to reply to the user's messages because it makes the chat seem more real.However, the send_message is chosen to make the interface clearer and cleaner since the robot will reply to many pieces of content at one time when replying to the weather.

The following is the result of two weather query functions.




current_weather result




forecast_weather result

# 4 Conclusion

In conclusion, this report explains some of the working principles of a usual chatbot and gives the process of implementing a simple weather query chatbot. The result shows that a good chatbot can help users do numbers of works which improves their work efficiency and in many other fields. It can be applied in various area and have great potential. It also shows that it is not easy to build a smart chatbot. There are still some limitations on how to improve the performance of the chatbot since rasa train may not be able to cover all the sentences in my case. With the development of techniques and scientists' effects, natural language analysis chatbot will become real smart in the near future.