

远程科研指导项目总结报告

聊天机器人项目总结

姓 名：姜才武

学 校：中山大学

专 业：智能科学与技术

目录

一、项目概述	3
二、背景知识	3
1. 自然语言处理	3
2. 正则表达式	3
3. Rasa NLU	3
三、过程分析	4
1. 获取数据	4
2. 制作训练集	5
3. Rasa 训练数据	6
4. 状态机的设置	6
5. 与微信集成	8
四、结果展示	8
1. 可用功能	8
2. 前端展示	9
五、总结	9

一、项目概述

该项目通过使用 Rasa 机器语言对话框、正则表达式、状态机思想以及 wxpy 的 API，构造了一个可以查询地区天气以及回答部分闲谈内容的系统。

利用这个系统，使用者可以快速、准确地得到特定城市的实时天气、天气预报和温度等信息。同时，该系统还具备一定的自然语言处理能力，在理解用户发送的语言之后可以提取出特定的信息并返回答案。

二、背景知识

该项目的背景知识主要包括三块，自然语言处理、正则表达式以及 Rasa、Spacy 的使用。

1. 自然语言处理 (Natural Language Processing)

自然语言，我的理解就是从人类传播的语言，比如汉语、英语、日语、德语等等。与之相对的就是编程语言等为计算机而设计的“人造”语言，也称作机器语言，就是计算机能理解的语言，比如 C、C++、Python 等等。

自然语言作为人类语言，包含着许多不确定性，所以基于句法-语义规则的理性主义方法受到了质疑，处理真实文本变成了自然语言处理的主要战略目标。其次词汇的作用在这个方向变得越来越重要，出现了所谓“词汇主义”的倾向。可以理解，当语料库越来越丰富，那么机器的应答就会越来越全面。

自然语言是人工智能最为困难的问题之一，所以研究这个方向也是富有挑战性的，但是却能为人们的生活带来肉眼可见的便利。

2. 正则表达式 (Regular Expression)

正则表达式是对字符串操作的一种逻辑公式，也就是用实现定义好的一些特定字符以及其组合组成一个“规定字符串”。

而通过 re 这个 package，我们可以实现以下功能：

1. 判断给定的字符串是否符合正则表达式的过滤匹配
2. 通过正则表达式取出字符串中我们需要的特定部分

总的来说，正则表达式的思想非常简单，但是上手却有点困难，它具有规范性并且晦涩难懂。但是熟练运用后会发现它有很强的灵活性、逻辑性和功能性，可以在自然语言处理方面发挥巨大的作用。

3. Rasa NLU

首先要说说 Rasa，Rasa 是一个用语构建智能聊天机器人的机器学习框架，它分为两部分，第一个是 NLU，第二个是 Core，NLU 主要负责的就是意图识别以及命名实体识别。Core 负责的则是把握住聊天并且决定下一步动作。在这里我们只需要用到 Rasa NLU。

简单地举一个例子，首先用 Rasa NLU 训练完一份数据后，尝试输入一句话：

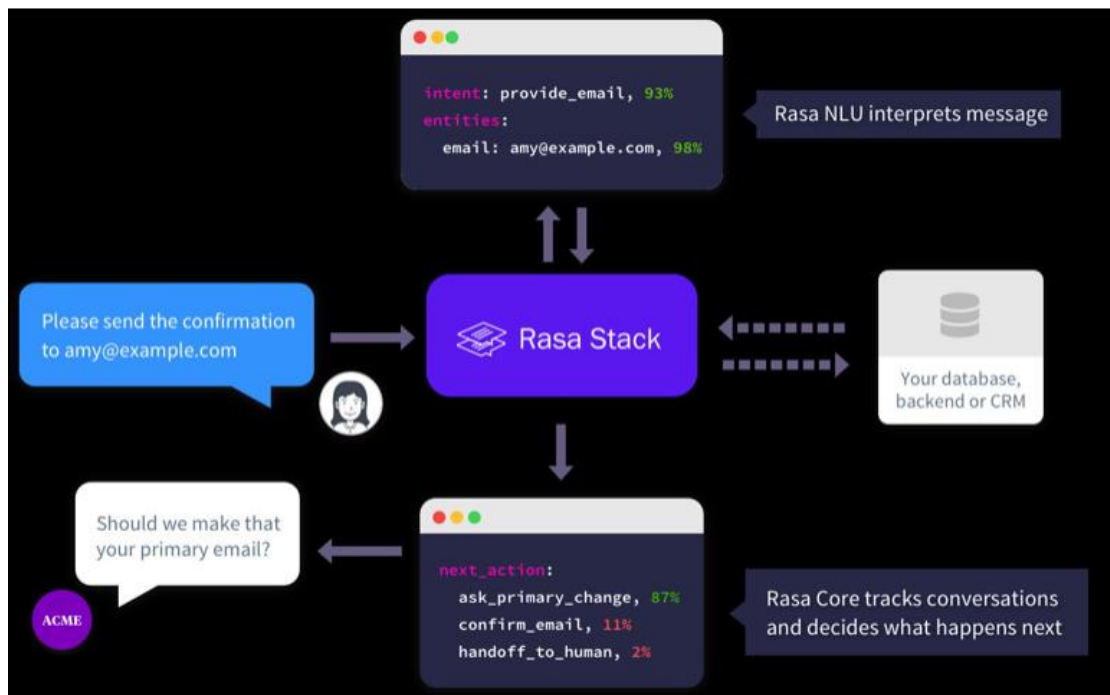
```
"I am looking for a Mexican restaurant in the center of town"
```

将会返回以下结果：

```
{
  "intent": "search_restaurant",
  "entities": {
    "cuisine": "Mexican",
    "location": "center"
  }
}
```

这种字典结构对于我们的代码编写及其清晰且方便。

它的使用流程很简单，首先编写一个 JSON 文件来放置语料用于训练，其次对上述文件进行训练，最后就可以得到训练的结果并将其特定部分提取出来用于语言的构造。



三、过程分析


整个过程分为以下几个部分：数据的获取、rasa 训练集的制作、状态机的设置、训练集的训练过程以及与微信的集成。

1. 获取数据

该项目所需的天气数据全由 [Open Weather Map](#) 来提供，该 API 提供了四种功能，如下图

GET Current Weather Data
GET Daily Forecast Weather Data
GET Forecast Weather Data
GET Search Weather Data

并且可以复制它的爬虫源代码，如下图：

```
Code Snippet (Python) Requests  Install SDK  
  
import requests  
  
url = "https://community-open-weather-map.p.rapidapi.com/weather"  
  
querystring = {"callback": "test", "id": "2172797", "units": "\"metric\" or \"imperial\"", "mode": "xml",  
html", "q": "London,uk"}  
  
headers = {  
    'x-rapidapi-host': "community-open-weather-map.p.rapidapi.com",  
    'x-rapidapi-key': "SIGN-UP-FOR-KEY"  
}  
  
response = requests.request("GET", url, headers=headers, params=querystring)  
  
print(response.text)
```

只需要下载 requests 包，并且将 querystring 中 “q” 的 value 改成想查询的城市名称，便可成功得到数据。

但是这些函数返回的是 str 类型的字符串而不是字典，因此对我们的提取过程有些阻碍。于是我先去除掉返回的字符串前面的 “test” 字符，留下后面的类字典字符串，再用 eval 函数将其转换成真正的字典，我们就能提取出想要的部分来构成回答，问题也就迎刃而解了。

```
response1=response.text  
s=response1[4:]  
b=eval(s)  
return [b['name'],b['weather'][0]['description'],b['wind']['speed']]
```

2. 制作训练集

经过在网上查资料，我明白训练集的格式可以有多种，我在这里使用的就是 JSON 训练集。它的格式其实也不难模仿，并且编辑时，如果格式不符合规范，编辑器并会立刻标红表示错误，这样让编写的难度大大降低。

训练集里面可以有 “entity_synonyms”，也就是同义词，可以让输入的容错率变高。因为是关于天气的训练集，所以内容主要围绕日常询问天气的语句来进行构建。在格式上主要分为三点，“text” 也就是训练的语句、“intent” 就是该句的意图、“entities” 就是句子中的实体。除此之外要注意的就是缩进与逗号的使用。具体格式如下：

```
{
  "text": "tell me about the weather in wuhan tomorrow.",
  "intent": "weather_forecast",
  "entities": [
    {
      "start": 28,
      "end": 33,
      "value": "wuhan",
      "entity": "city"
    },
    {
      "start": 34,
      "end": 42,
      "value": "tomorrow",
      "entity": "time"
    }
  ]
},
]
```

3. Rasa 训练数据

前面已经将 JSON 格式的训练集构造完毕，之后就是训练过程：

```
# Import necessary modules
from rasa_nlu.training_data import load_data
from rasa_nlu.config import RasaNLUModelConfig
from rasa_nlu.model import Trainer
from rasa_nlu import config

# Create a trainer that uses this config
trainer = Trainer(config.load("/home/jcw/rasa_nlu/sample_configs/config_spacy.yml"))

# Load the training data
training_data = load_data('/home/jcw/rasa_nlu/data/examples/rasa/weather-rasa.json')

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

其中的 trainer 是一个训练机，内容为

```
language: "en"

pipeline: "spacy_sklearn"
```

“language”项中的“en”指的是语言类型：英语，因为该项目主要针对英文对话。而“pipeline”我们选择 spacy_sklearn 是因为我们使用的是 Spacy 的 en_core_web_md 语料库。

4. 状态机的设置

而为了实现多轮查询，我运用了状态机的思路，如下图：

```

#Define the states
INIT=0
AUTHED=1
ASK_ABOUT_WEATHER=2
# Define the policy rules
policy_rules = {
    (INIT,"none"):[(INIT, "you'll have to log in first, what's your phone number?", AUTHED)],
    (AUTHED,"none"):[(AUTHED,"what do you mean?",None),(AUTHED,"Sorry, I can't help you.",None),(AUTHED,"You can ask me something about weather.",None)],
    (ASK_ABOUT_WEATHER,"none"):[(ASK_ABOUT_WEATHER,"what do you mean?",None),(ASK_ABOUT_WEATHER,"Sorry, I can't help you.",None)],
    (INIT,"ask_explanation"):[(INIT, "I'm a clever weather robot and I can provide you with a lot of data about weather.",None)],
    (AUTHED,"ask_explanation"):[(AUTHED, "I'm a clever weather bot and I can provide you with a lot of data about weather.",None)],
    (ASK_ABOUT_WEATHER,"ask_explanation"):[(ASK_ABOUT_WEATHER, "I'm a clever weather bot and I can provide you with a lot of data about weather.",None)],
    (INIT,"ask_usage"):[(INIT, "1.I can search weather for you. Try to ask me in this way:\nI want to know the current weather.",None)],
    (AUTHED,"ask_usage"):[(AUTHED, "1.I can search weather for you. Try to ask me in this way:\nI want to know the current weather.",None)],
    (ASK_ABOUT_WEATHER,"ask_usage"):[(ASK_ABOUT_WEATHER, "1.I can search weather for you. Try to ask me in this way:\nI want to know the current weather.",None)],
    (INIT,"number"):[(AUTHED, "perfect, welcome back!", None)],
    (AUTHED,"number"):[(AUTHED,"Sorry, I can't help you.",None),(AUTHED,"You can ask me something about weather.",None)],
    (ASK_ABOUT_WEATHER,"number"):[(ASK_ABOUT_WEATHER,"Sorry, I can't help you.",None),(AUTHED,"You can ask me something about weather.",None)],
    (INIT,"number"):[(AUTHED, "perfect, welcome back!", None)],
    (AUTHED,"number"):[(AUTHED,"Sorry, I can't help you.",None),(AUTHED,"You can ask me something about weather.",None)],
    (ASK_ABOUT_WEATHER,"number"):[(ASK_ABOUT_WEATHER,"Sorry, I can't help you.",None),(AUTHED,"You can ask me something about weather.",None)],
    (INIT,"service_require"):[(INIT, "I can help you, but you have to log in first, so what's your phone number?", AUTHED)],
    (AUTHED,"service_require"):[(AUTHED, "Well, please tell me exactly what do you want to search?", AUTHED)],
    (ASK_ABOUT_WEATHER,"service_require"):[(ASK_ABOUT_WEATHER,"Well, please tell me exactly what do you want to search?", AUTHED)],
    (INIT,"current_weather"):[(INIT, "you'll have to log in first, what's your phone number?", AUTHED)],
    (INIT,"weather_forecast"):[(INIT, "you'll have to log in first, what's your phone number?", AUTHED)],
    (INIT,"temperature_search"):[(INIT, "you'll have to log in first, what's your phone number?", AUTHED)],
    (AUTHED,"current_weather"):[(ASK_ABOUT_WEATHER, "OK, The current weather in {} is mainly {}, with the force-{}.",None)],
    (ASK_ABOUT_WEATHER,"current_weather"):[(ASK_ABOUT_WEATHER, "OK, The current weather in {} is mainly {}, with the force-{}.",None)],
    (AUTHED,"current_weather"):[(ASK_ABOUT_WEATHER, "OK, The current weather in {} is mainly {}, with the force-{}.",None)],
    (ASK_ABOUT_WEATHER,"current_weather"):[(ASK_ABOUT_WEATHER, "OK, The current weather in {} is mainly {}, with the force-{}.",None)],
    (AUTHED,"weather_forecast"):[(ASK_ABOUT_WEATHER, "n", None)],
    (ASK_ABOUT_WEATHER,"weather_forecast"):[(ASK_ABOUT_WEATHER, "n",None)],
    (AUTHED,"temperature_search"):[(ASK_ABOUT_WEATHER, "In {}, the max temperature is {}°C, the min temperature is {}°C.",None)],
    (ASK_ABOUT_WEATHER,"temperature_search"):[(ASK_ABOUT_WEATHER, "In {}, the max temperature is {}°C, the min temperature is {}°C.",None)],
    (INIT, "goodbye"):[(INIT, "goodbye!",None),(INIT, "See you next time!",None)],
    (AUTHED, "goodbye"):[(INIT, "goodbye!",None),(INIT, "See you next time!",None)],
    (ASK_ABOUT_WEATHER, "goodbye"):[(INIT, "goodbye!",None),(INIT, "See you next time!",None)],
    (INIT, "greet"):[(INIT, "Hello!",None),(INIT, "I'm glad to see you.",None),(INIT, "Hi!",None),(INIT, "Welcome here!",None)],
    (AUTHED, "greet"):[(AUTHED, "Hello!",None),(AUTHED, "I'm glad to see you.",None),(AUTHED, "Hi!",None),(AUTHED, "Welcome here!",None)]
}

```

为了防止出现不存在的情况，我列举出了所有可能的状态与意图之间的搭配。

为了记录所有的回答，在最终的 sendmessage 函数中引入 answer 变量，函数如下图：

```

def send_message(state, pending, message, answer):
    print("USER : {}".format(message))
    response = chitchat_response(message)
    if response is not None:
        answer=response
        print(answer)
        return state, None, answer

    # Calculate the new_state, response, and pending_state
    new_state, response, pending_state = random.choice(policy_rules[(state, interpret(message))])
    if interpret(message) == 'current_weather':
        city=interpreter.parse(message)['entities'][0]['value']
        (a,b,c)=(current_weather(city)[0],current_weather(city)[1],current_weather(city)[2])
        answer=response.format(a,b,c)
    elif interpret(message) == 'weather_forecast':
        city=interpreter.parse(message)['entities'][0]['value']
        answer=weather_forecast(city)
    elif interpret(message) == 'temperature_search':
        city=interpreter.parse(message)['entities'][0]['value']
        (a,b,c,d)=(temperature_search(city)[0],temperature_search(city)[1],temperature_search(city)[2],temperature_search(city)[3])
        answer=response.format(a,b,c,d)
    else:
        answer=response

    if pending is not None and new_state == pending[0]:
        new_state, response, pending_state = random.choice(policy_rules[pending])
        answer=response
        print(answer)
    if pending_state is not None:
        pending = (pending_state, interpret(message))
    print(answer)
    return new_state, pending, answer

```

并且为了记住上下文的内容，我创建了一个类用来记录，如下图：

```
class StateMachine(object):
    def __init__(self):
        self.state = 0
        self.record = None
        self.pending = None
        self.answer = 0
        print(self.state, self.pending, self.answer, self.record)
stateMachine = StateMachine()
```

5. 与微信集成

这块主要参考 [wxpy: 用 Python 玩微信](#), 首先要用一个帐号登录 wechat bot 作为机器人

```
#Log in the wechat bot
from wxpy import *
bot = Bot()
```

```
Getting uuid of QR code.
Downloading QR code.
Please scan the QR code to log in.
```

系统会弹出一个二维码，扫码登陆之后该帐号将作为聊天机器人，而其他帐号想要触发该帐号的聊天机器人的功能，必须先得到该帐号的授权

```
a = bot.friends().search("###")[0]
@bot.register(a)
def auto_reply(msg):
    state = INIT
    pending = None
    answer = 0
    state, pending, answer = send_message(state, pending, msg.text, answer)
    return answer
```

上图中的@bot.register(a)，就是对 a 的授权。

而 wxpy 中还有许多有趣的功能，可以尝试去实现，这里就不再赘述。

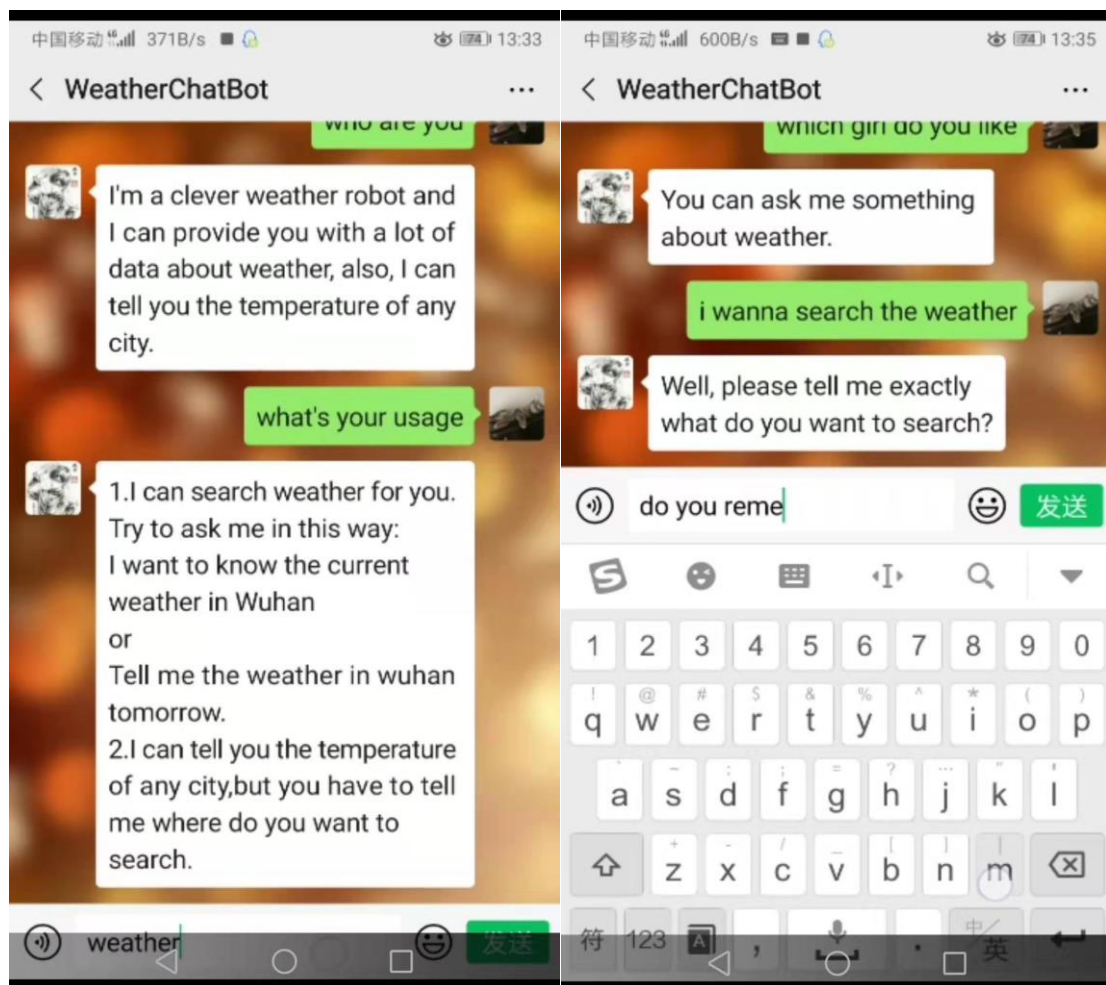
四、结果展示

1. 可用功能

该项目所需的天气数据全由 [Open Weather Map](#) 来提供，该 API 提供了四种功能，如下图

意图	含义	示例
goodbye	与机器人再见	goodbye/farewell/end
greet	与机器人打招呼	hi/hello/good morning
ask_explanation	询问当前功能的解释	What can you do for me?
ask_usage	询问机器人用法	how can I ask question?
number	号码	555-3624
service_require	服务请求	Can you help me search the weather?
current_weather	查询某地点的实时天气	I want to know guangzhou's weather.
weather_forecast	查询某地点的天气预报	how is the weather in guangzhou tomorrow?
temperature_search	查询某地点的最高温、最低温以及平均温度	show me the temperature in wuhan.
none	无法理解的意图	which girl do you like?

2. 前端展示



五、总结

本次的远程科研指导项目使我收获颇丰。以前经常听到同学提起 NLP 问题，但自己却从未真正去了解过，只是单纯知道它的英文全称以及中文含义罢了，而这次的项目给了我一个自己动手处理 NLP 问题的机会。

在本次项目开始之前，我并没有学过 python，只学过 C 与 C++。而我用这次项目中的一个月的时间在 python 这块有了极大的进步，并编写出了一个像模像样的聊天机器人。

在项目的过程中，老师讲授了自然语言的发展以及应用的场景、正则表达式的应用、意图识别、命名实体识别、词向量、支持向量机、否定、多轮查询、状态机等等知识。这其中有难有易，有些容易理解一看就懂比如正则表达式、意图识别，而有些则需要花时间去思考，比如词向量、支持向量机以及否定的使用。这些知识点分散开来容易复现，可是要把它们结合在一起并且最终连贯使用却又是一个难点。

再来仔细说说我的实验过程。我本来打算利用 iexfinance 实现一个金融查

询机器人，奈何它的许多 API 接口无法使用，我认为是因为我在国内的问题，于是尝试购买外国的服务器来顺利使用 API，但是无奈 Linux 系统的翻墙实在太过困难，折腾了一天的我只能在 Rapidap.com 网站上另寻他法。经过斟酌，我发现一个好用的 API，Open Weather Map，于是我决定以天气为主题，实现一个可以查询各地天气的智能聊天机器人。

因为用到了爬虫，所以得到的格式为字符串，这使得它原有的字典功能失去了效用。于是我在网上查询，找到了解决办法，先将字符串前面的四个字母 test 用提取的方法去掉，再用 eval 函数将其转化为字典。做到这里我不禁感叹 python 的强大，这要是用换作 c 那真是难以下手。

解决了这个问题之后就是状态机的设置以及训练集的制作了。因为有以前别的主题的 JSON 格式的训练集，我也就照葫芦画瓢写出了一个天气相关的训练集，虽然不能包括所有的询问方式，但应该也是够用了。除了天气相关，我还在训练集中加了一些打招呼的数据以及一些无关数据，这是为了可以使训练出来的机器人理解能力和应答能力更加全面。状态机的设置也不容易，我想了好几张情况，最终决定还是只需要三种状态，应答主要由识别出的意图来决定，并且在状态机中引入 pending，方便准备下一次应答。为了不出现键为状态与意图的值不存在的情况，我的 policy rules 写了很长一串。

这之后就使如何将我之前提取到的数据与应答结果结合到一起的过程了，这一步让我想了好一会儿。因为 wxpy 中的 bot 应答是用到 return 的，这就意味着之前要将 answer 保存下来，于是我在 send_messages 函数中添加了一个 answer 变量，成功地解决了这一问题。还有就是 pending 不等于 None 时的消息保存问题，这让我再加了一个 record 变量来专门记录。

在与微信集成的过程中，我还遇到一个问题，就是难以记录前后文的内容，于是我尝试建立一个类，让该类里面的变量来记录每次生成的特定内容，在运用到下一次应答生成过程中，结果非常成功

整个过程做完后，在测试的过程中也会逐渐发现问题，比如状态机用不了，原因是 JSON 训练集中出现了拼写错误，识别出来之后在状态机中无法匹配。也出现了意图的识别偏差问题。不断的测试能让我不断地对其进行优化，虽然还是不能达到完美，但是应该也够用了。

总的来说，其实过程不是那么顺利，遇到了许多的低级问题。很多代码只有自己写过才知道，我也是在不断的修改中逐步加深印象，逐渐加深对 NLP 问题的理解。

最后，我想说这次项目令我收获颇丰，实实在在地学到了许多硬核的、实际的知识，也同样要感谢老师这一个月辛苦教学，给我在聊天机器人这块打了一块好的基础，让我能逐渐窥探到其中的奥秘。