# 远程科研报告

人工智能方向

2019.01.08

# 目录

一、研究背景介绍..... 03
二、知识点回顾..... 03
三、项目详细介绍..... 10
四、其他内容..... 13
五、技术难点..... 14
六、总结与思考..... 17

# 一、研究背景介绍

聊天机器人（Chatterbot）是经由对话或文字进行交谈的计算机程序。可用于信息交互，智能查询，语音输入等实际用途。目前，聊天机器人常作为虚拟助理的一部分，作为增加产品用户体验或智能感的营销手法应用于各类科技产品中。

目前比较被大众接受的应用产品可大体分为两类，第一类主要作为智能助手，协助用户以更简单的方式传递并执行自己的意图，丰富人们的生活方式。人们使用这类虚拟助手时，通常以实现生活需求为主，如语音查询航班信息，路况信息，预定酒店，自动设定闹钟等。大部分智能助手不需要单独的设备，通常都被内置于手机、电脑、音响等常见的电子设备中，为人们广泛接受。

相关产品包括人们较为熟知的Siri，Echo 智能音箱，Google Home。用户可通过语音输入或文字输入传递给智能助手信息，接着，智能助手会扮演在网络中搜索相关数据的媒介，实施汇报查询结果。通过调用多种开放的API获得大量数据。

另一大类，则是被广泛使用的智能客服咨询机器人和智能导览介绍系统。这种方式以智能聊天程序替代了重复率较高的介绍活动，节约人力成本，是目前人工智能应用的典型代表之一。客服机器人原理与本项目实现的查询机器人十分类似，功能上主要强调多轮对话技术，根据需求给予用户合适的信息，一定程度上具有记忆功能。达美乐、必胜客、迪士尼、Nerdify、雅玛多 Line、全食超市都已推出各自的聊天机器人，以便与终端消费者增进交流，推销公司的产品与服务，并且让消费者订货更加方便。2016年，观光业的一些旅行社和航空公司透过 Messenger 推出了聊天机器人的服务，墨西哥航空利用人工智能售票、回答问题，墨航和荷兰皇家航空并且提供航班资讯，处理乘客报到，发出行动登机证，推荐旅馆、餐厅、目的地行程。

此外，其他用途的聊天机器人还有很多，比如语音遥控的智能家居，单纯用于闲聊功能的手机应用程序：simsimi，和以微软小冰，索菲亚为代表的高级人工智能等。

在这样的背景下，为了进一步拓展知识技能，对目前的自然语言处理、聊天机器人技术进行学习和应用。本项目通过现有的机器学习技术方法，基于rasa训练数据，结合python语言，实现了在微信平台上运行的智能聊天助手。该智能助手在功能上包含一般闲聊和实时数据查询两个部分。同时，为了在有限时间内对聊天机器人技术知识点做出全面的了解和学习， 项目将重心放在了代码的可行性和自然语言处理的知识的应用上。因而弱化了操作界面的搭建，选用wxpy便捷地将聊天智能助手整合在微信上运行。

# 二、知识点回顾

## 1. 正则表达式

正则表达式是对字符串操作的一种逻辑公式，多用于字符串的检索，替换，项目中被多次运用在文本中提取关键信息。

### 1.1 模式匹配

判定输入信息是否符合特定形式

```
import re    #载入python中正则表达式相关模块re
pattern = “Do you remember.*”
#定义送入信息规则，所有满足“Do you remember…”开头的信息都会被识别出
message = “Do you remember when you last watched the movie?”
match= re.search(pattern, message)
#在整个字符串中搜索匹配规则，满足则返回第一个匹配，否则返回None
if match:
    print(“string matches!”)
```

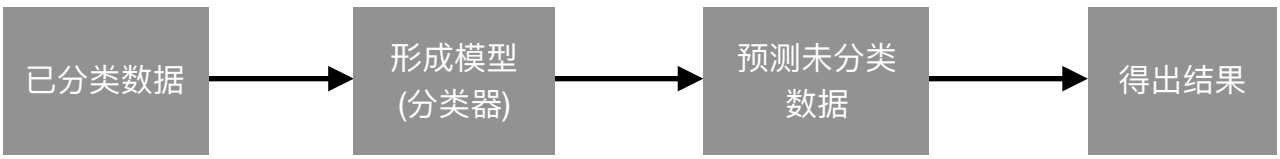
### 1.2 提取关键词：抽取出字符串里待替换的信息

```
import re
pattern= r'if(.*)'
message ="what would happen if german team lost the world cup?"
match = re.search(pattern, message)
match.group(0)      #if german team lost the world cup?
match.group(1)      #german team lost the world cup?
```

### 1.3人称转换：将抽取出的字符根据需求进行替换，产生回复信息

```
def replace_pronouns(phase):  #phase为上一步抽取出的信息
   if "I" in phase:
      phase = re.sub("I","you".phase)
   if "my" in phase:
      phase = re.sub("my","your",phase)
   else:
      return phase
   return phase
```

## 2. 监督学习




监督学习是指：利用一组已知类别的样本调整分类器的参数，使其达到所要求性能的过程。通过分类器预测给定的句子的意图标签，通过预测的准确度来评估测试数据的性能。数据越多数据质量越高分类器识别准确度越高。

训练数据：通过一定量的已经带有标签的数据训练模型

准确度：预测正确的标签数在所有被预测的数据中的占比

# 3. 词向量

把每一个单词转化为向量，转化为向量后可以求词语的cos值。通过cos值的大小得出两个词语的相关度。相同语义的词具有相似的向量，出现在上下文中的词具有相似的向量。通过这两个原则，在海量的文本库中将各个词语建立联系。项目中使用的词语向量库为300维的向量库：en_core_web_md

cos值为1:词语的向量指向相同的方向，语义高度相关
cos值为0:词语向量垂直，语义无相关
cos值为-1:词语向量方向相反，语义完全相反

”Can you help me please.”
向量表示：“can”, ”you”, “help”, “me”, “please”

## 3.1 安装spacy向量库

```
import spaCy
nlp = spaCy. load(‘en_core_web_md’)
nlp = vocal.vectors_length
```

## 3.2 词语相似度判断:similarity()函数

```
import spaCy
nlp = spaCy. load(‘en_core_web_md’)
doc = nlp(“cat”)
doc,similarity(nlp(“can”))   #out:0.30165
doc,similarity(nlp(“dog”))   #out:0.80168
```

通过结果可以判断，“cat”与“dog”的cos值更接近1，语义相似；但“cat”与“can”的cos值接近0，虽然拼写相近，但语义相似度较低。

# 4. 意图识别和最近邻分类法

以ATIS航空数据集为例，ATIS航空数据集是从真实航空数据中搜集，成千上万带有标记意图和实体的句子，适合作为训练数据使用。以数据中前两个句子为例，通过代码调取数据集中前两个句子：

```
sentences_train[:2]
```

Out:[“I want to fly from Boston at 838 am and arrive in Denver at 1110 in the morning”,”what flight are available from Pittsburgh to Baltimore on Thursday morning”]

同样用代码调取这两个句子的标签,从而得知这两个句子的意图都是在询问合适的航班：

```
labels_train[:2]
```

Out:[“atis_flight”,”atis_flight”]

当要判断一个新的不存在于库中的句子的意图时，把新的句子也变成300维的向量，并和已经存在于矩阵中的句子相比较，通过计算cos值进行判断，得出相似度最高的句子，新句子的意图则和相似度最高的句子一致。这是一种无监督学习方法——最近邻分类法。代码如下：

```
import numpy as np
X_train_shape = (len(sentences_train), nlp.vocab.vectors_length)
#建立一个X_train_shape矩阵，矩阵长度为训练句子的长度，宽度是300维
X_train = np.zeros(x_train_shape)
#把矩阵全部赋值为0，记为X_train
for sentence in sentences_train:
    X_train[i:] = nlp(sentence).vector
#获得每个句子的向量，并赋值进X_train
```

## 5. SVM/SVC 支持向量机

支持向量机是一种比最近邻分类法效率更高的方法，但结果的准确性相较于最近邻分类法略低一点，大部分情况下不影响结果。原理是，将训练数据集中的句子分类，每个类别有一个语义中心。新来的句子和类别中心比较即可。

```
from sklearn. svm import svc
clf = svc()    #定义一个分类器
clf.fit(x_train, y_train)   #训练出分类器模型
y_pred = clf.predict(x_test)  #预测新句子的意图
```

## 6. 实体抽取

### 6.1 预建的实体识别: 项目中使用的是spaCy中预建的实体识别

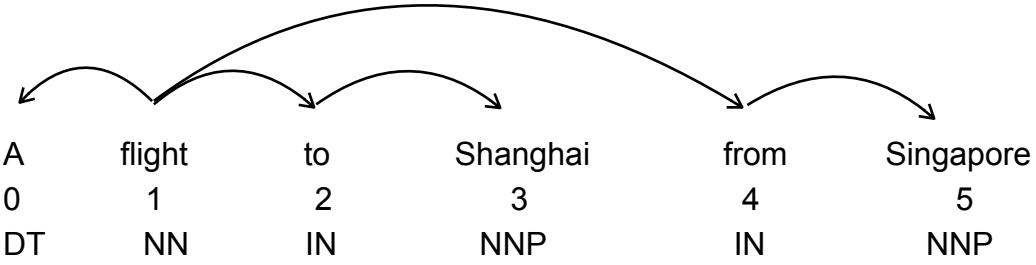
```
import spacy
nlp = spacy.load("en_core_web_md")
doc = nlp("my friend Marry has worked at google since 2009.")
for ent in doc.ents:
    print(ent.text, ent.lable_)

out: Marry PERSON
      google ORG
      2009 DATE
```

### 6.2 定义句式

以ATIS数据集中航班查询的句子为例：

```
I want a flight from Tel Aviv to Bucharest
Show me the flight to Singapore from Shanghai

pattern-1 = re.compile(r'.*from(.*)to(.*)')
pattern-2 = re.compile(r'.*to(.*)from(.*)')
```

6.3 依赖分析法




```
doc = nlp(“a flight to Singapore from Shanghai”)
sh, sg = doc[3],doc[5]
list(sh. ancestors)              #通过ancestors参数找到依赖和修饰关系
out:[to, flight]                 #通过“to”判断出Shanghai是目的地

list(sg. ancestors)
out:[from, flight]               #通过“from”判断出Singapore是出发地

NNP(proper None)
DT(Determiners)
IN(prepositions and subording conjunctions )
NN(common Noun)
```

# 7. RASA NLU

## 7.1 使用rasa训练数据

```
from rasa_nlu.training_data import load_data
Training_data= load_data(“PATH/TO/training_data.json”)]
Import json
print(json.dumps(training_data.entity_examples[0].data, intent=2))

Out:{“intent”：”restaurant_search“，“entity”:[{“start”:31,
                                                “end’:36,
                                                “value”:”north”,
                                                “entity”:”location” }
                                               ]}
```

## 7.2 翻译器：解析信息, 识别意图

```
message = “I want to book a flight to London”
Interpreter.parse(message)

Out:{
“intent”:{
     “name”:”flight_search”,
     “confidence”:0.9},
 “entities”:[
         {“entity”:”location”,
          “value”:”London”,
          “start”:27,
          “end”:33 } ]  }
```

### 7.3 rasa的使用

```
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

trainer = Trainer(config.load("PATH/TO/CONFIG_FILE"))    #建立一个训练器
training_data = load_data('PATH/TO/TRAINING_DATA')    #load进训练数据
interpreter = trainer.train(training_data)    #通过训练器对数据的训练建立一个解析器
```

## 8. API和数据库

在使用人工智能虚拟助手时，为了实现某些实际的功能，比如查询餐厅，酒店等。需要获得真实存在的数据，有两种获取方式。第一是使用API接口直接获取，第二是建立本地数据库。API的使用部分下文中"项目详细内容"板块有应用例，这里主要对数据库知识做回顾。

### 8.1 基本SQL语句

```
SELECT*from restaurants;    #检索下表所有内容并返回
SELECT name, stars from restaurants;      #只搜索name&stars栏并返回
SELECT name from restaurants WHERE location = "center" AND price = "hi";
#搜索同时满足location为center，price为hi两个条件的餐厅并返回餐厅名字
```

| name | price | location | stars |
|---|---|---|---|
| Bills Burgers | hi | east | 3 |
| Moe's Plaice | lo | north | 3 |
| Sushi Corner | mid | center | 3 |

### 8.2 python中SQLite的使用

```
import sqlite3
conn = sqlite3.connect("hotel.db")
c = conn.cursor()
c.execute("SELECT*FROM hotels WHERE location = 'south' and price = 'hi' ")

c.fetchall()

Out:[("Grand Hotel","hi","south",5)]
```

## 9. 多轮查询

```
#通过params参数进行记录
def respond(message, params):
    #更新消息中实体的参数
    #运行查询
    #挑选回应
return response, params

params={ }
response, params = respond(message, params)
```

## 10. 否定实体

我们通过“not”和“n’t”进行否定，在实际使用时将每个句子拆分成很多子句，在字句中查找”not“和“n’t”。

```
                0   1   2    3       4    5
def = nlp（“not sushi，maybe pizza?”)
   indices = [1,4]                    #1和4分别对应sushi和pizza
   ents, negated_ents = [ ],[ ]     #建立实体和否定实体的列表
   start = 0

   for i in indices:         phrase = “{}”.format(doc[start:i])
       if “not” in phrase or “n’t” in phrase:
            negated_ents.append(doc[i])
       else:
            ents.append(doc[i])
       start = i
#分段遍历把带有not或者n’t的实体存入negated_ents，其余实体存入ents
```

## 11. 有状态的聊天机器人

没有状态的机器人：前后的对话为了实现同一个目的

step1：请问母亲的电话号码　#要号码意图
step2：请把电话打给他　　#打电话意图

有状态的机器人：

step1:我们喜欢无状态的系统
step2:他们有缺点吗　　#他们指代无状态系统，隐含了状态信息
step3:难道没有缺点吗

### 11.1 状态机 (state machine)

定义三个状态:
INIT = 0
CHOOSE_COFFEE = 1
ORDERED = 2

定义一个字典类型的状态跳转规则，字典的键是包含当前状态和当前意图的二元组，值是包含新状态和回复内容的二元组。

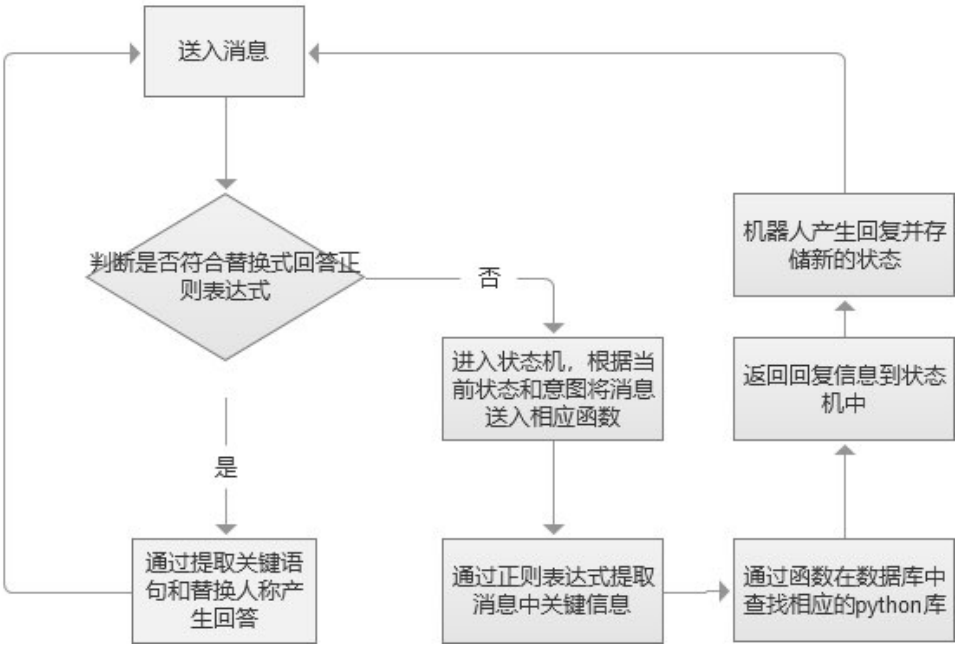
```
policy_rules = {
 (INIT,”order”):(CHOOSE_COFFEE.”ok, columbian or kenyan?”),
 (CHOOSE_COFFEE,”specify_coffee”):(ORDERED.”perfect, the beans are on their way!”),
}
```

### 11.2 使用状态机

```
state = INIIT

def respond(state, message):
            (new_state, response) = policy_rules[(state, interpret(message))]\
            return new_state, response

def send_message(state, message):
           new_state, response = respond(state, message)
           return new_state

state = send_message(state, message)
```

# 三、项目详细介绍

## 1. 简要框架



## 2. 程序详细介绍

API： 通过api接口获得相关数据并把相关的信息存储为字典

```
#通过request获取Github上关于python库的实时信息/API
import requests
url = "http://api.github.com/search/repositories?q=language:python&sort=stars"
r = requests.get(url)
response_dict = r.json()
repo_dicts = response_dict ["items"]


#通过 iexfinance进行股票信息的相关查询
def get_price(message):
    nlp = spacy.load("en_core_web_md")
    doc = nlp(message)

    if doc.ents == ()
        return "sorry, I can't find any information.”
    else:
        ent = doc.ents[0].text
        price = Stock(ent)
        price.get_open()
        price.get_price()

        batch = Stock([ent])
        return "The price of {} is {}. ".format(ent, batch.get_price())


def get_data(message):
    nlp = spacy.load("en_core_web_md")
    doc=nlp(message)
    if doc.ents == ():
        #ent = None
        return "sorry, I can't find any information."

    else:
        ent=doc.ents[0].text
        start = datetime(2019, 1, 13)
        end = datetime(2019, 1, 17)
        df = get_historical_data(ent, start=start, end=end, output_format='pandas')
        return df.head()
```

## 3.功能函数

### 3.1 通过名字查询所有信息

通过python仓库的名字查询该仓库的详细信息，输出内容包括仓库名字，作者信息，更新时间等，具体回复如下：

information - Name: sentry, Owner: getsentry, Stars: 19880, Repository: https://github.com/getsentry/sentry Created: 2010-08-30T22:06:41Z, Updated: 2019-02-25T15:51:25Z, Destription: Sentry is cross-platform application monitoring, with a focus on error reporting.

## 3.2 通过单个信息查询

通过单个信息，如仓库的url地址、作者名字等反向查询仓库的名字，获得仓库名字后，可继续追问获得该仓库的其余详细信息。程序通过列表分别对每一条新送入的消息，新计算出的状态进行记录，再结合RASA训练好的意图识别器对最近一条消息的意图进行判断，匹配相应的执行函数。这里以用过作者名字查询仓库信息为例，代码如下：

```
#policy_rules中部分信息
(START_SEARCH, "get_item_owner"): (SEARCH_OWNER, get_item_owner(message)[0]),
(SEARCH_OWNER, "otherinfor"):(START_SEARCH,other_infor(get_item_owner(message)[1])),

#通过owner获得project名字
def get_item_owner(message):
    pattern = re.compile(r'-(.*?)-')   #从message 中提取owner
    i = 0
    match = re.search(pattern, message)
    if match is None:
        return "Sorry, no repository is eligible.",None
    if match is not None:
        owner = match.group(1)
    while i < 30:
        if str(owner) == repo_dicts[i]["owner"]["login"]:
            return  random.choice(answer).format(repo_dicts[i]['name']),i
        else:
            i+=1
    return "Sorry, no repository is eligible",None

#查询其余信息
def other_infor(msg):
    if msg is None:
        return "Sorry, I can't find any information."
    else:
        i = int(msg)
        return "information - Name: {}, Owner: {}, Stars: {}, Repository: {} Created: {}, Updated: {},
Destription: {} ( ^ — ^ )".format(repo_dicts[i]['name'],repo_dicts[i]["owner"]["login"],repo_dicts[i]
["stargazers_count"],repo_dicts[i]["html_url"],repo_dicts[i]["created_at"],repo_dicts[i]
["updated_at"],repo_dicts[i]["description"])
```

## 3.3 查询全部仓库数量

```
#获取github实时python库的数量
def find_python_total_count():      return "There are {0} repositories of Python on Github. I can
show you statistic chart of them If you want .".format(response_dict["total_count"])
```

## 3.4 查询最受欢迎的仓库及其信息

```
#查询最受欢迎的item
def find_top_item():
    return "The most popular repository is: {0}.".format(repo_dicts[0]['name'])
```

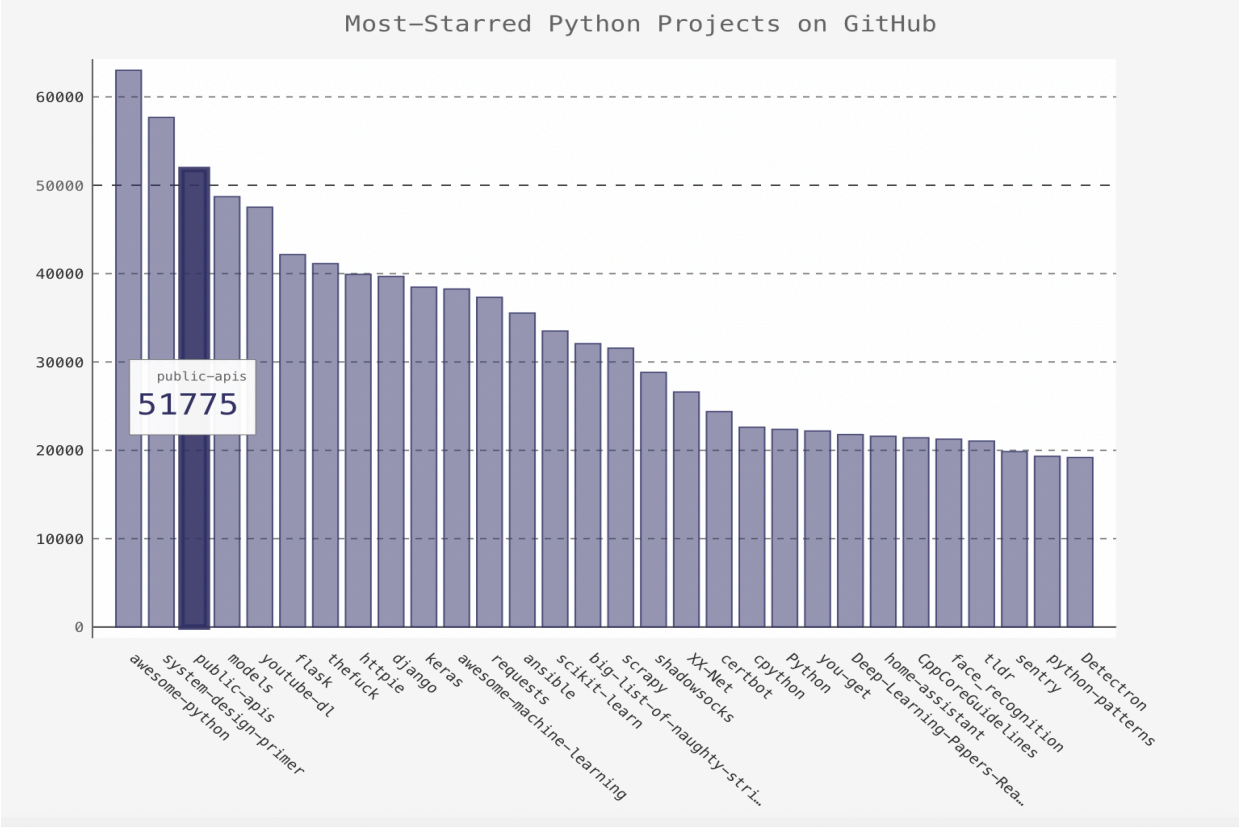
# 四、其他内容

## 1.数据可视化

为了进一步拓展和应用项目中学到的知识，也为了更直观的获得python库的数据，我尝试用以下代码将通过API获得的数据做可视化，代码直接运行会构建一个可交互的svg文件，如下图所示，当鼠标悬停时会自动显示当前Python库的名字和Stars数量。

```
import pygal
from pygal.style import LightColorizedStyle as LCS, LightenStyle as LS

names, stars = [] , []
for repo_dict in repo_dicts:
    names.append(repo_dict["name"])
    stars.append(repo_dict["stargazers_count"])

my_style = LS("#333366", base_style = LCS)
chart = pygal.Bar(style = my_style, x_label_rotation = 45, show_legend = False)

chart.title = "Most-Starred Python Projects on GitHub"
chart.x_labels = names

chart.add(" ",stars)
chart.render_to_file(“python_repos.svg”)
```

结果：



# 五、技术难点

## 1.policy_rules中的函数递送信息

与课程中的应用例不同，实现项目时，机器人的回复不是固定的消息，而是根据用户请求查询的结果，因此如何将用户送入的消息及时更新到policy_ruiles中是我一开始没有解决也没有考虑到的问题。在测试中发现这个问题时，我意识到产生这样的问题原因在于message信息改变时，程序只记录了新的意图，但没有及时的将新的message送入policy_rules中，而导致policy_rules调用查询函数时出现参数错误。

对此，我采取的办法是，在送入新的信息之后，对policy_rules中包含message参数的键值对进行重新定义。以保证查询函数中的message参数始终是最近获得的消息。当然这样的处理办法在一定程度上增加了程序整体的复杂度，导致运行速度有所下降。后期优化时，我将需要重新定义的内容按照意图进行分类，每一个意图分类中包含最多六条重新定义的键值对，大大减小了每次需要重新定义的内容，一定程度上增加了程序效率。部分代码如下：

```
intent = interpret(message) #对消息意图进行识别

#根据意图将相应message内容及时替换进policy_rules
if intent == "get_stock_price":
    policy_rules[(START_SEARCH, "get_stock_price")] = (STOCK_PRICE, get_price(message))

elif intent == "name":
   policy_rules[(INIT, "name")] =
   (INIT,random.choice(responses['name']).format(get_entity(message)))

elif intent == "get_stock_history_data":
   policy_rules[(STOCK_PRICE, "get_stock_history_data")] =
   (START_SEARCH, get_data(message))
   policy_rules[(START_SEARCH, "get_stock_history_data")] =
   (STOCK_DATA, get_data(message))
   policy_rules[(SEARCH_URL, "get_stock_history_data")] = (STOCK_DATA, get_data(message))
```

## 2.抽取实体信息

为了增加聊天机器人的灵活性，需要从多种消息中抽取供查询的关键词。但是由于Github上用户的名字和python库的名字变化非常大，也没有明显的特征，可以说是五花八门，没有规律可言。因此不能通过传统的实体识别的方法抽取关键词。为了有效的解决这个问题，我决定采用准确但没有太多变化空间的“ 正则表达式” 来提取关键词。也就出现了机器人对话中：“if you want to search by NAME, remenber using # # to highlight it. if you want to search by owner,please using - - to highlight the owner's name :) ”这样的提示。当然在项目结束后，我对这个问题做了进一步的思考，因为正则表达式属于一种比较死板的方法，而上文中所提到的依赖分析法，可以较为灵活的解决这个问题。

## 3.使用wxpy后新状态的保存问题

这个问题的主要原因是在调试整个程序的过程中，一直是以循环的方式送入信息直接产生多轮问讯和回答，以便及时发现问题。但在最后要将程序通过wxpy整合到微信上时，通过for循环语句连续送入的信息变成了单次送入的信息，从而导致不能通过循环语句直接记录程序执行后获得的新状态。原代码如下：

```
# Send the messages
messages = [
  'what can you do?',
   "yes",
   "find a repository called #system-design-primer#",
   "How many python rerepositories on github?",
   "no",
   "can you tell me a joke",
   "show me the most popular repository!",
   "give me other details",
   "find a repository called #system-design-primer#",
   "No I don't like this "
]

# Define send_messages()
def send_messages(messages):
   state = INIT
   pending = None
   pending_state = None
   for msg in messages:
      state, pending = send_message(state, pending, msg)

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

   # Calculate the new_state, response, and pending_state
   intent = interpret(message)
   new_state, response, pending_state = policy_rules[(state, intent)]
   print("BOT : {}".format(response))
   if pending is not None:
      new_state, response, pending_state = policy_rules[pending]
      print("BOT : {}".format(response))
   if pending_state is not None:
      pending = ( interpret(message),pending_state)
   return new_state, pending
```

由以上代码可以看出，原程序主要是通过send_messages()和send_message()两个函数嵌套循环来获得新的状态，但在wxpy的机制中每一条新消息只能通过register调用一个函数一次，每调用一次函数产生一次回复信息，获得的结果不会再次作为参数送入下一次该函数的调用中。为了将程序运行一次后产生的新状态记录下来，我将消息和状态分别定义为两个列表，每次送入的信息会被添加到列表中，通过主程序计算出的新状态也会被添加进列表。以这种处理方式，在通过register调用函数时可以直接从列表中获取最近的状态，实现了新状态的记录：

```
messages = ["0"]
states = [INIT]
message = messages[-1]

from wxpy import *
bot = Bot(cache_path = True)
myFriend = bot.friends("keep going") #确定接收消息的对象，“keep going”为自动回复对象的微信ID

@bot.register(chats= None, msg_types=None, except_self=True) #注册消息处理方法：自动回复
def forward_message(message):
    state = states[-1]
    messages.append(message.text)
    answer = send_message(state,messages[-1])
    return answer

embed()
```

## 4.调用不同函数时状态的跳转

为了增加程序的鲁棒性，使得程序在任何对话阶段下都能实现合理的回复，梳理policy_rules中状态变化的逻辑十分必要。比如，在询问url的多轮对话情景下插入对python库owner的名字的查询时，机器人仍然能实现正常的状态跳转。只要不断有信息送入，就能一直实现各个状态的切换。为了实现这一功能，重点是实现状态循环的跳转，并在所有状态下设置好退出查询和进入查询的通路。部分代码如下：

```
(START_SEARCH, "find_information_name"):
(SEARCH_NAME,find_information_name(message)),
(SEARCH_NAME, "find_information_name"):
(START_SEARCH,find_information_name(message)),
(SEARCH_NAME, "get"): (START_SEARCH, "Always happy to help you!"),
(SEARCH_NAME, "deny"): (START_SEARCH, "Sorry, I can't find other relative information. Hope to help you again :)"),
(SEARCH_NAME, "get_random"): (START_SEARCH, random_item()),
(SEARCH_NAME, "thankyou"): (START_SEARCH, "My pleaseure."),
(SEARCH_NAME, "get_item_owner"): (SEARCH_OWNER, get_item_owner(message)[0]),
(SEARCH_NAME, "get_item_url"): (SEARCH_URL, get_item_url(message)[0]),
(SEARCH_NAME, "get_item_star"): (SEARCH_STARS, get_item_star(message)[0]),
```

# 六、总结与思考

以这次项目为契机，我有幸全面地学习了目前业界广泛使用的聊天机器人程序使用的方法和自然语言处理的相关知识。不但精进了python相关技能，培养了自学能力，也引发了进一步的思考。

为了完成项目，我做了大量相关资料的阅读，以构建相关领域知识的基础认识，熟悉相关概念，主要通过利用网络资源查找相关案例，和使用《Deep Learning With Python》一书。我认为要快速地在一个领域内入门，首先第一步就是通过阅读资料使得自己在这一领域建立起一个大框架的认识，激发自身兴趣，站在巨人的肩膀上。以便在遇到瓶颈时，及时找到解决问题的大方向，然后再顺着这个方向去学习相关的解决办法。人工智能和自然语言处理属于比较复杂的知识范畴，初次尝试项目，处处碰壁不可避免，但感谢老师的帮助，这些问题都一一解决了。

就代码本身而言，我一边编写项目程序，一边学习了《Python Crash Course》一书，在编写项目时我对书中给出的知识和案例进行了尝试，在项目中有所体现，比如API的获取和数据可视化。此外，每一章的代码讲解和课后练习题，也充分地协助了我理解课程中所涉及到的代码。这些学习和练习使我对python开发更加熟悉，也非常可观地培养了我的自学能力和知识总结能力。

就项目整体而言，本次项目的目的以学习为主，重点考虑如何成功地将学习到的知识点应用出来，注重功能的实现和代码的完整性。在优化的方向上，也主要是在程序算法的简洁性，程序的结构和易读性上做了提升。因而没有过多的考虑软件的用户体验的部分，项目中所选用的Github数据资料查询和股票资料查询相对来说受众较少，希望在找到合适的API或数据资料后对项目功能进行进一步的完善和提升。

在接触这个项目之前，我自身从事比较多的领域是数字媒体技术应用和虚拟现实技术方面，我认为虚拟现实技术和人工智能技术的结合是一个非常有潜力的交叉领域，也正是因为对这个领域的好奇和探索欲望使得我产生了参加本次项目的动机。因为目前的人工智能技术发展虽然遇到一些限制但已经足够在大众生活中的很多方面做出改变，目前人工智能被探索的应用领域多在教育方向和工业方向上。但这两类方向都较容易受到实体设备的限制，而且制作成本高昂，难以推广。相对而言虚拟现实技术已经发展的比较完善，如果能开发相关的增强现实应用程序，不仅可以大大降低成本，也可以使得人工智能技术真正的走入大众生活。一个伪全息投影的智能助手会比实体的智能机器人更加方便也更加节约成本，人工智能算法也可以让已有的虚拟现实产品功能更加人性化。

通过这次项目我不但实现了实际参与人工智能项目，了解相关知识的初衷，也对自己最初的设想有了一些新的认识，四周的学习虽然只是对这一领域的窥探但确实为我的知识技能树提供了新的方向，相信在日后的开发和应用程序构思中我能有更多的思路和见解，对研究生阶段的知识选择也会有更好的判断。