MIT 麻省理工大学远程科研项目之

智能对话机器人项目

科研报告

林天成

2020.9-2020.10

目录

一、研究背景

1.1 自然语言处理历史背景

自然语言处理大体是从 1950 年代开始，虽然更早期也有作为。1950 年，图灵发表论文“计算机器与智能”，提出现在所谓的“图灵测试”作为判断智能的条件。

1954 年的乔治城实验涉及全部自动翻译超过 60 句俄文成为英文。研究人员声称三到五年之内即可解决机器翻译的问题。不过实际进展远低于预期，1966 年的 ALPAC 报告发现十年研究未达预期目标，机器翻译的研究经费遭到大幅削减。一直到 1980 年代末期，统计机器翻译系统发展出来，机器翻译的研究才得以更上一层楼。

1960 年代发展特别成功的 NLP 系统包括 SHRDLU——一个词汇设限、运作于受限如“积木世界”的一种自然语言系统，以及 1964-1966 年约瑟夫·维森鲍姆模拟“个人中心治疗”而设计的 ELIZA——几乎未运用人类思想和感情的消息，有时候却能呈现令人讶异地类似人之间的交互。“病人”提出的问题超出 ELIZA 极小的知识范围之时，可能会得到空泛的回答。例如问题是“我的头痛”，回答是“为什么说你头痛？”

许多早期的成功属于机器翻译领域，尤其归功 IBM 的研究，渐次发展出更复杂的统计模型。这些系统得以利用加拿大和欧盟现有的语料库，因为其法律规定政府的会议必须翻译成所有的官方语言。

近来的研究更加聚焦于非监督式学习和半监督学习的算法。这种算法，能够从没有人工注解理想答案的资料里学习。大体而言，这种学习比监督学习困难，并且在同量的数据下，通常产生的结果较不准确。不过没有注解的数据量极巨（包含了万维网），弥补了较不准确的缺点。

1.2 自然语言处理的难点

单词的边界界定[编辑]

在口语中，词与词之间通常是连贯的，而界定字词边界通常使用的办法是取用能让给定的上下文最为通顺且在文法上无误的一种最佳组合。在书写上，汉语也没有词与词之间的边界。

词义的消歧[编辑]

许多字词不单只有一个意思，因而我们必须选出使句意最为通顺的解释。

句法的模糊性[编辑]

自然语言的文法通常是模棱两可的，针对一个句子通常可能会剖析（Parse）出多棵剖析树（Parse Tree），而我们必须要仰赖语义及前后文的信息才能在其中选择一棵最为适合的剖析树。

有瑕疵的或不规范的输入[编辑]

例如语音处理时遇到外国口音或地方口音，或者在文本的处理中处理拼写，语法或者光学字符识别（OCR）的错误。

二、研究过程

2.1 同一个问题多种选择性的回答，并提供缺省回答的方案;

(1) 代码思路：在这里我们主要考虑通过一个字典来存储不同的问题的不同答案，每一个问题关键字作为一个 key，对应多个答案，用 random 模块来进行随机选择

(2) 代码实现：

```
bot_template = "BOT : {0}"
user_template = "USER : {0}"
import random
responses = {'statement': ['tell me more!',
                           'why do you think that?',
                           'how long have you felt this way?',
                           'I find that extremely interesting',
                           'can you back that up?', 'oh wow!', ':)'],
             'question': ["I don't know :(",
                          'you tell me!']
            }

def respond(message):
    # Check for a question mark
    if message.endswith("?"):
        # Return a random question
        return random.choice(responses["question"])
    # Return a random statement
    return random.choice(responses["statement"])

# Define a function that sends a message to the bot: send_message
def send_message(message):
    # Print user_template including the user_message
    print(user_template.format(message))
    # Get the bot's response to the message
    response = respond(message)
    # Print the bot template including the bot's response.
    print(bot_template.format(response))
```

(3) 输出结果：

```
USER : what's today's weather?
BOT : I don't know :(
USER : what's today's weather?
BOT : you tell me!
USER : I love building chatbots
BOT : I find that extremely interesting
USER : I love building chatbots
BOT : oh wow!
```

2.2 能通过正则表达式、模式匹配、关键词提取、句法转换等来回答问题;

(4) 代码思路：在这里我们引入了一个正则表达式模块，并且学习了正则表达式相关的知识，思路是通过部分缺省的正则表达式来匹配问题，也就是进行模式匹配，成功匹配时就将 match_group 提取出来，并进行句法转换，将人称代词转换，最后输出。

(5) 代码实现：

```
import re
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'],
         'if (.*)': ["Do you really think it's likely that {0}",
                     'Do you wish that {0}',
                     'What do you think about {0}',
                     'Really--if {0}']
        }

# Define match_rule()
def match_rule(rules, message):
    response, phrase = "default", None

    # Iterate over the rules dictionary
    for k in rules.keys():
        # Create a match object
        match =re.search(k,message)
        if match is not None:
            # Choose a random response
            response = random.choice(rules[k])
```

```
                if '{0}' in response:
                    phrase = match.group(1)
    # Return the response and phrase
    return response, phrase
```

```
def replace_pronouns(message):

    message = message.lower()
    if 'me' in message:
        # Replace 'me' with 'you'
        return re.sub('me','you',message)
    if 'my' in message:
        # Replace 'my' with 'your'
        return re.sub('my','your',message)
    if 'your' in message:
        # Replace 'your' with 'my'
        return re.sub('your','my',message)
    if 'you' in message:
        # Replace 'you' with 'me'
        return re.sub('you','me',message)

    return message
```

```
def respond(message):
    # Call match_rule
    response, phrase = match_rule(rules, message)
    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
```

(6)输出结果:

```
USER : do you remember your last birthday
BOT : Why haven't you been able to forget my last birthday
USER : do you think humans should be worried about AI
BOT : if humans should be worried about ai? Absolutely.
USER : I want a robot friend
BOT : What would it mean if you got a robot friend
USER : what if you could be anything you wanted
BOT : Do you wish that me could be anything me wanted
```

2.3 能通过正则表达式、最近邻分类法或者支持向量机之一或多种方案来提取用户意图；

(7) 代码思路：在这里我们主要考虑先创建关键词字典，然后根据每个意图设置几个不同的关键词，再将它们使用 re.compile 函数变为正则表达式，这样就可以进行匹配问题中的关键词，来提取用户的意图，之后在创建 responses 字典，根据不同的意图来进行回复。

(8) 代码实现：

```
import re
keywords = {
                    'seasons': ['season', 'th', 'year'],
                    'teams': ['team', 'thx'],
                    'players': ['he', 'him','his'],
                    'games': ['game', 'date'],
                   }#创建字典
# Define a dictionary of patterns
patterns = {}

# Iterate over the keywords dictionary
for intent,keys in keywords.items():
    # Create regular expressions and compile them into pattern objects
    patterns[intent] = re.compile("|".join(keys))
```

```
responses = {'seasons': 'Wow ,that is a wonderful season!',
                    'teams': 'haha,i know you like this team',
                    'players': 'So you want something about him?',
                    'games': 'you wanna check the details of that game?'
                   }

# Define a function to find the intent of a message
def match_intent(message):
    matched_intent = None
    for intent, pattern in patterns.items():
        # Check if the pattern occurs in the message
        if re.search(pattern,message):
            matched_intent = intent
    return matched_intent

# Define a respond function
def respond(message):
    # Call the match_intent function
```

```
    intent = match_intent(message)
    # Fall back to the default response
    key = "default"
    if intent in responses:
        key = intent
    return responses[key]
```

(9) 输出结果：

```
USER : season
BOT : Wow ,that is a wonderful season!
USER : him
BOT : So you want something about him?
USER : date
BOT : you wanna check the details of that game?
```

2.4 通过预建的命名实体类型、角色关系、依赖分析等来进行命名实体识别；

(10) 代码思路：在这里我们引入了 spacy 模块，并且下载他自带的一个 en 语言模块，根据 en 中的实体属性，我们就可以进行实体抓取，并且可以给抓取到的实体有不同的标签，例如 include_entities = ['DATE', 'ORG', 'PERSON']，抓取到的实体自动会有这三个标签中的一个，来便于我们的后续工作。

(11) 代码实现：

```
def extract_entities(message):
    # Create a dict to hold the entities
    ents = dict.fromkeys(include_entities)
    # Create a spacy document
    doc = nlp(message)
    for ent in doc.ents:
        if ent.label_ in include_entities:
            # Save interesting entities
            ents[ent.label_] = ent.text
    return ents

text1='The player you want is {0}'.format(conn.request("GET", "/teams/city/Cleveland", headers=headers))
text2='The team you want is {0}'.format(extract_entities('friends called Lebron james who have worked at Google since 2010')['ORG'])
text3='The season you want is {0}'.format(extract_entities('friends called Lebron james who have worked at Google since 2010')['DATE'])

print(extract_entities('friends called Lebron james who have worked at
```

```
Google since 2010'))
print(text1)
print(text2)
print(text3)
```

(12) 输出结果:

```
{'DATE': '2010', 'ORG': 'Google', 'PERSON': 'Lebron james'}
The player you want is Lebron james
The team you want is Google
The season you want is 2010
```

2.5 基于 Rasa NLU 的本地基础聊天机器人系统的构建

(13) 代码思路：在这里我们首先安装 rasanlu，这个模块可以进行意图识别和实体识别以及实体属性分析，但是需要首先定义数据集并且进行训练。

(14) 代码实现:

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

# Load the training data
training_data = load_data('demo-rasa.json')

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

# Try it out
print(interpreter.parse("I'm looking for a player called lebron"))
```

(15) 输出结果:

```
Fitting 2 folds for each of 6 candidates, totalling 12 fits
{'intent': {'name': 'restaurant_search', 'confidence': 0.7557001821568343},
'entities': [], 'intent_ranking': [{'name': 'restaurant_search', 'confidenc
e': 0.7557001821568343}, {'name': 'greet', 'confidence': 0.0684317636696257
2}, {'name': 'affirm', 'confidence': 0.06529717729407698}, {'name': 'goodby
e', 'confidence': 0.06053391723472587}, {'name': 'hotel_search', 'confidenc
e': 0.031452282218862}, {'name': 'location', 'confidence': 0.01858467742587
517}], 'text': "I'm looking for a player called lebron"}
```

2.6 数据库查询并使用自然语言探索数据库内容（提取参数、创建查询、响应）;

(16) 代码思路：在这里我们引入数据库模块，首先我们连接一个数据库，然后给数据库输入数据，接下来应用 c.execute 来进行数据库命令的实现。

(17) 代码实现:

```
import sqlite3

# Open connection to DB
conn = sqlite3.connect('hotels.db')

# Create a cursor
c = conn.cursor()

# Define area and price
location, price = "east", "mid"
t = (location, price)

# Execute the query
c.execute('SELECT * FROM hotels WHERE location=? AND price=?', t)

# Print the results
print(c.fetchall())
```

(18) 输出结果:

```
[('Hotel for Dogs', 'mid', 'east', 3)]
```

2.7 基于增量过滤器的单轮多次增量查询技术以及甄别否定实体技术;

(19) 代码思路：在这里我们使用 params 列表来进行参数存储，每次进行 respond 的时候还会将之前的参数和 message 一起输入作为新的参数，一次来进行增量过滤，并且达到单论多次对话的效果。

(20) 代码实现：

```
Define a respond function, taking the message and existing params as input
def respond(message, params):
    # Extract the entities
    entities = interpreter.parse(message)["entities"]
    # Fill the dictionary with entities
    for ent in entities:
        params[ent["entity"]] = str(ent["value"])

    # Find the hotels
    results = find_hotels(params)
    names = [r[0] for r in results]
    n = min(len(results), 3)
    # Return the appropriate response
    return responses[n].format(*names), params

# Initialize params dictionary
params = {}

# Pass the messages to the bot
for message in ["I want an expensive hotel", "in the north of town"]:
    print("USER: {}".format(message))
    response, params = respond(message, params)
    print("BOT: {}".format(response))
```

(21) 输出结果：

```
USER: I want an expensive hotel
BOT: Grand Hotel is one option, but I know others too :)
USER: in the north of town
BOT: Bens BnB is a great hotel!
```

2.8 实现状态机的多轮多次查询技术，并能基于语境问题提供解释和回答；

(22) 代码思路：在这里我们首先定义不同的状态的代表数字，并且进行 policy 规定，即规定不同的状态检测到不同的关键词时的状态跳转，这样就可以通过状态的转换来完成多轮多次的对话。

(23) 代码实现：

```
def send_message(policy, state, message):
    print("USER : {}".format(message))
    new_state, response = respond(policy, state, message)
    print("BOT : {}".format(response))
    return new_state

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

def interpret(message):
    msg = message.lower()
    if 'order' in msg:
        return 'order'
    if 'kenyan' in msg or 'columbian' in msg:
        return 'specify_coffee'
    return 'none'
```

```
# Define the policy rules
policy = {
    (INIT, "order"): (CHOOSE_COFFEE, "ok, Colombian or Kenyan?"),
    (INIT, "none"): (INIT, "I'm sorry - I'm not sure how to help you"),
    (CHOOSE_COFFEE, "specify_coffee"): (ORDERED, "perfect, the beans are on their way!"),
    (CHOOSE_COFFEE, "none"): (CHOOSE_COFFEE, "I'm sorry - would you like Colombian or Kenyan?"),
}

# Create the list of messages
messages = [
    "I'd like to become a professional dancer",
    "well then I'd like to order some coffee",
    "my favourite animal is a zebra",
    "kenyan"
]
```

```
# Call send_message() for each message
state = INIT
for message in messages:
    state = send_message(policy, state, message)
```

(24) 输出结果：

```
USER : I'd like to become a professional dancer
BOT : I'm sorry - I'm not sure how to help you
USER : well then I'd like to order some coffee
BOT : ok, Colombian or Kenyan?
USER : my favourite animal is a zebra
BOT : I'm sorry - would you like Colombian or Kenyan?
USER : kenyan
BOT : perfect, the beans are on their way!
```

2.9 处理拒绝、等待状态转换和待定行动的多轮多次查询技术；

(25) 代码思路：在这里我们根据不同的关键词来判断用户的意图，比如说如果检测到 no 就时 deny 的意图，如果检测到 yes 就是 affirm 的意图，根据不同的意图做不同的状态跳转或者执行待定的选项。

(26) 代码实现：

```
def policy(intent):
    # Return "do_pending" if the intent is "affirm"
    if intent == "affirm":
        return "do_pending", None
    # Return "Ok" if the intent is "deny"
    if intent == "deny":
        return "Ok", None
    if intent == "order":
        return "Unfortunately, the Kenyan coffee is currently out of stock, would you like to order the Brazilian beans?", "Alright, I've ordered that for you!"
```

```
def interpret(message):
    msg = message.lower()
    if 'order' in msg:
        return 'order'
    elif 'yes' in msg:
        return 'affirm'
    elif 'no' in msg:
        return 'deny'
```

```
        return 'none'
def send_message(pending,message):
        print("USER : {}".format(message))
        action, pending_action = policy(interpret(message))
        if action == "do_pending" and pending is not None:
                print("BOT : {}".format(pending))
        else:
                print("BOT : {}".format(action))
        return pending_action

# Define send_messages()
def send_messages(messages):
        pending_action = None
        for msg in messages:
                pending_action = send_message(pending_action, msg)

# Send the messages
send_messages([
        "I'd like to order some coffee",
        "ok yes please"
])
```

(27) 输出结果:

```
USER : I'd like to order some coffee
BOT : Unfortunately, the Kenyan coffee is currently out of stock, would you
like to order the Brazilian beans?
USER : ok yes please
BOT : Alright, I've ordered that for you!
```

三、研究成果

3.1 自主机器人概述

这次我自主完成的机器人是一个 NBA 查询机器人，用到的 API 是 rapidapi 中的 NBA-API，这个 API 可以完成球员信息，赛季信息等信息的寻找，同时界面使用的是 telegram，为了将课程中学习到的代码嵌入，我们使用了 telegram-python 模块来进行后端的编写。使用的语言是 python。

3.2 前期准备

首先，我在 telegram 中与 botfather 对话来创建属于自己的机器人，拿到 token 之后就可以进行相关操作。

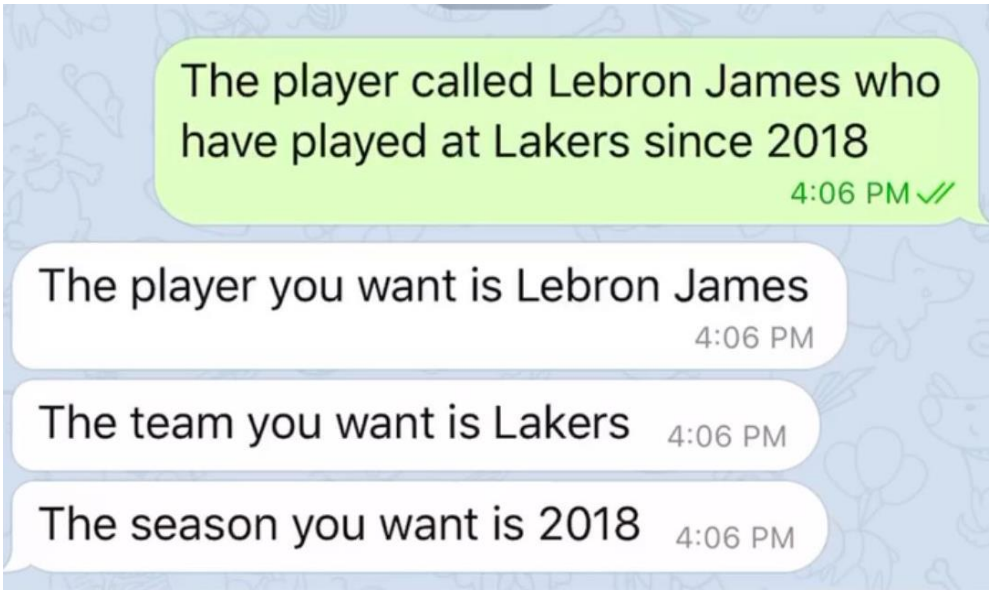
3.3 实现目标

目前希望实现的是多轮多次对话，共四轮，四个意图的对话，主要的意图有
(1) 查询球员
(2) 查询赛季
(3) 查询球队
(4) 查询具体比赛
每个意图的对话至少在两轮，具体实现的效果将展示在视频中，源代码因为和之前列举的代码高度相似因此就不再重复，具体请参见附件中的 NBArobot.ipynb。
除此之外，我还进行了多个单独的录像来展示关于课程不同部分内容的实现，再次简单列举。

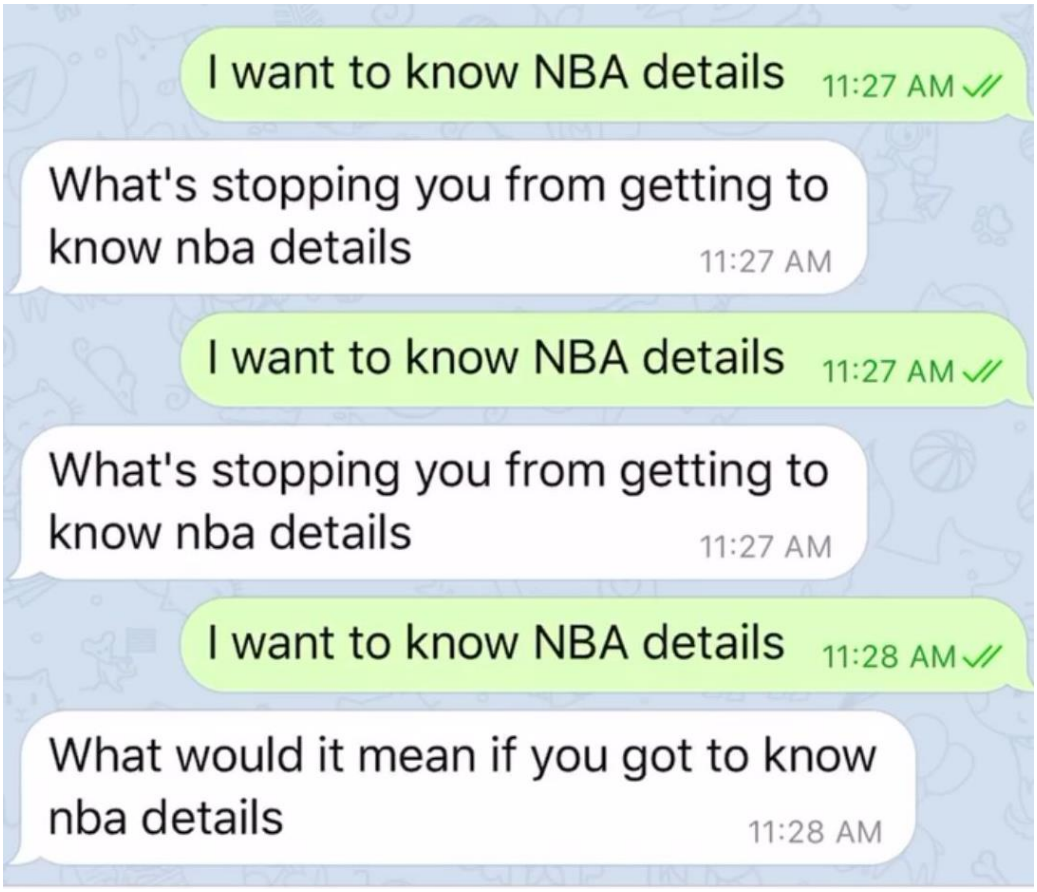
3.4 意图识别

3.5 同个命令的多种回复并提供缺省回答:

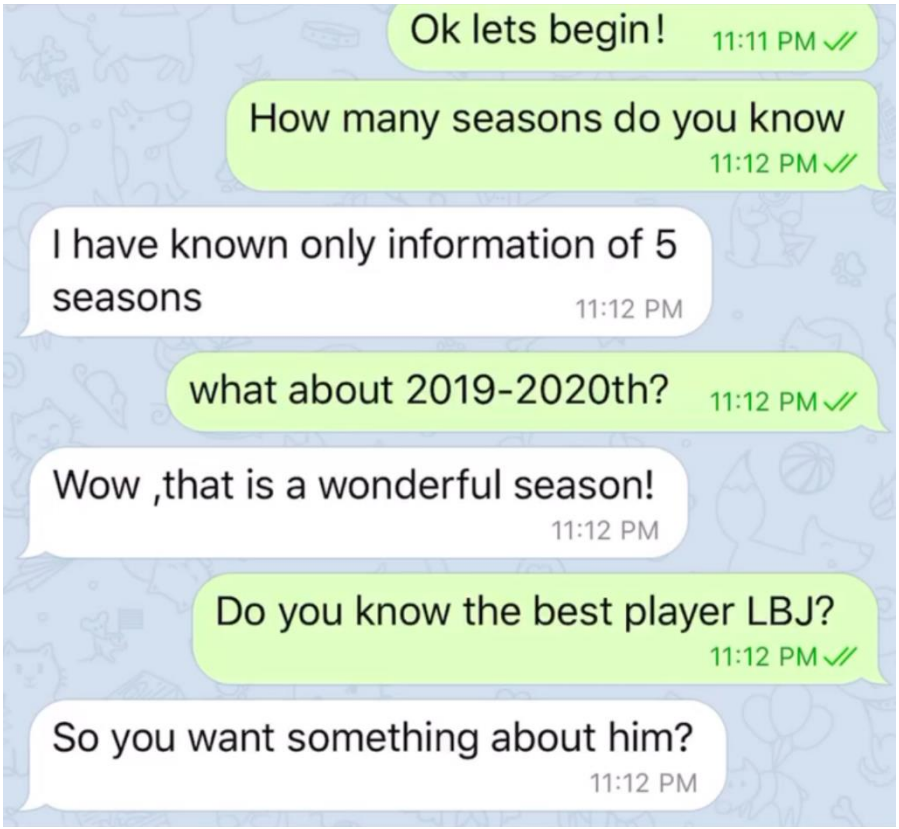
3.6 实体抓取:

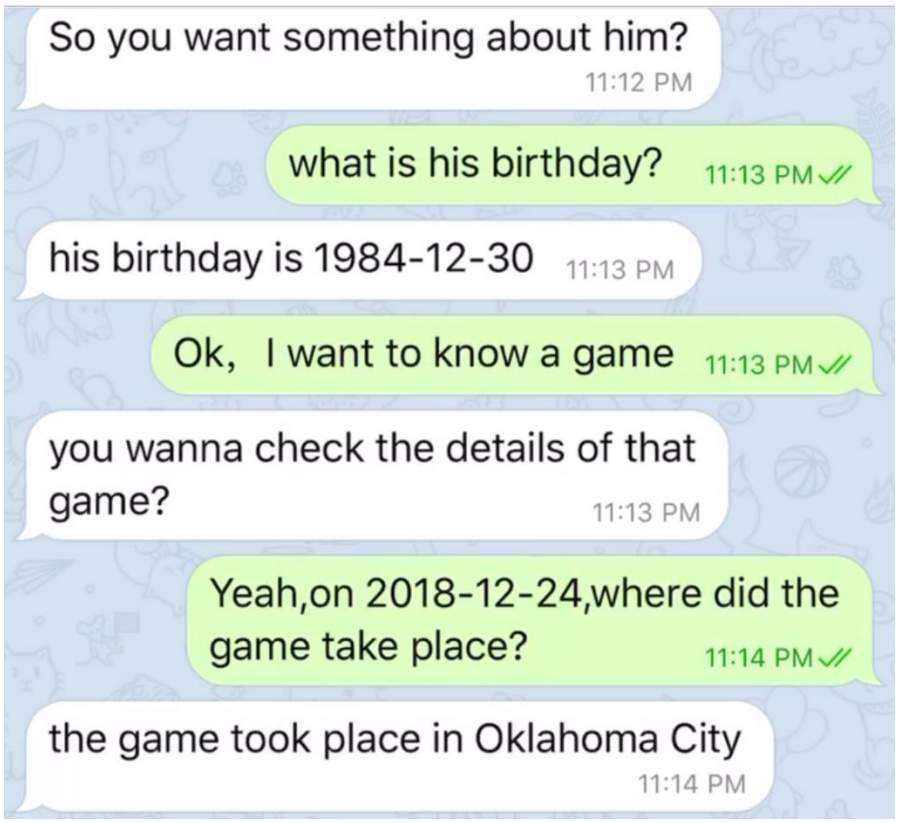


3.7 根据正则表达式进行模式识别并替换人称代词:



3.8 多轮多次对话:


Ok lets begin! 11:11 PM
How many seasons do you know 11:12 PM
I have known only information of 5 seasons 11:12 PM
what about 2019-2020th? 11:12 PM
Wow ,that is a wonderful season! 11:12 PM
Do you know the best player LBJ? 11:12 PM
So you want something about him? 11:12 PM


So you want something about him? 11:12 PM
what is his birthday? 11:13 PM
his birthday is 1984-12-30 11:13 PM
Ok, I want to know a game 11:13 PM
you wanna check the details of that game? 11:13 PM
Yeah,on 2018-12-24,where did the game take place? 11:14 PM
the game took place in Oklahoma City 11:14 PM


1
Chats
travisbot
bot
T
Ok, I want to know a game 11:13 PM
you wanna check the details of that game? 11:13 PM
Yeah,on 2018-12-24,where did the game take place? 11:14 PM
the game took place in Oklahoma City 11:14 PM
Finally,do you know anything about the team Cleveland? 11:14 PM
haha,i know you like this team 11:14 PM
Can you sent me their logo? 11:15 PM


1
Chats
travisbot
bot
T
Cavs_de_Cleveland_logo_2017.png
Today
CLEVELAND
CAVALIERS
11:15 PM
Message

四、项目总结

4.1 困难解决

在这次的项目过程中，大体上遇到了几个困难

（1） 对于 python 语言的不熟悉，缺乏练习导致的低级错误

（2） 对于 re 正则表达式的表达形式不熟悉导致的错误

（3） 对于 rasanlu 安装部署相关的各项错误

（4） 关于 telegram 网络相关的错误

解决方案则有：

（1） 对于 python 语言多加训练，在课前预习代码内容，争取能填的都填一遍，不会填的在课上针对性的听老师讲。

（2） 学习 re 相关内容，并在项目中进行训练和熟悉。

（3） 最终的解决方案是在一个虚拟环境中完成了安装，对于缺少的 wheel 要在资源网站上单独安装，要灵活使用 youtube，google 上面的资源来解决问题。

（4） 非常惭愧。因为这个问题卡了两天，最后的解决方案是使用 google colab 在云端进行编码，非常感谢老师的耐心帮助，不过也因祸得福，了解了这个十分有效的多人合作工具。

4.2 项目总结与收获

在这次为期两个月的项目中，我收获很多，在前一个月的项目课程中，我感受到了张帆导师极大的耐心和专业性，每次课的足足三个小时中我们都是几乎在讲代码，理论知识很少，非常注重实践，因此在 12 个小时的课程之后我的 python 水平有了很大的提高，并且对于自然语言处理中需要用到的相关模块有了具体的了解，对于机器学习在自然语言处理领域的应用也有了清晰的认识，在学习中，我尽量在每节课之前先把代码填空完成一遍，然后在课上老师讲解的时候再把不会的地方仔细听一下，下课后再复习一下视频资料，这样的学习模式非常有效，确实让我把难以理解的知识点都巩固的比较好。

而在后一个月的自主实验环节，我则犯上了拖延的毛病，第一周几乎都在整理之前课上的代码，因为不相信自己可以真的在一个对话软件上部署一个这样的机器人而畏手畏脚，再加上遇到了一些网络上的问题，每次在 google 搜索出的全英文界面我都有点抗拒，于是就进展缓慢，但是在老师耐心的问题解答之后，我也下定决心克服这种畏难情绪，跟着 github 给出的 instruction 一点一点部署机器人，根据课上老师给我们讲解的代码一点点修改成 telegram-python 中可以使用的代码，其实这个过程比我想象的要简单不少，只要开始做了，其实就克服了最难的一步。非常感谢这次课程的所有助教老师，主导师，和各位同学们，大家能够在课上课下帮助我克服困难，真的非常感谢！因为其实我一直不是一个非常善于做科研的人，对于科研和项目有一种强烈的畏难情绪，也是逼着自己看到新的代码就要先填写一

遍，这样才一点点跟上课程的进度，感谢老师的耐心讲解，和助教老师的悉心帮助，这次的项目让我受益匪浅，不仅仅是在技术层面上理解自然语言处理的关键和诀窍，更是从信心上一个很大的助力，让我相信自己也可以完成像样的机器人。
到这里这次的项目接近尾声了，其实从头到尾都没能跟各位见上一面，甚至视频都没开过，或许也不会有机会见面，但是感谢各位带给我的收获，也希望各位之后的路程一帆风顺！再见!