智能聊天机器人项目

总结报告

陈蔡鑫

2020.8.1

目录

一、 研究背景……3

二、 研究过程及结果分析……5

三、 学习感受与收获……15

一、 研究背景

随着科技的发展，越来越多的智能产物逐渐进入人们的生活，机器逐渐替代了更多种类的手工劳动，这都得益于人工智能学科的发展与创新。人工智能是研究、开发用于模拟、延伸和扩展人的智能的理论、方法、技术及应用系统的一门新的技术科学。

人工智能是计算机科学的一个分支，它企图了解智能的实质，并生产出一种新的能以人类智能相似的方式做出反应的智能机器，该领域的研究包括机器人、语言识别、图像识别、自然语言处理和专家系统等。人工智能从诞生以来，理论和技术日益成熟，应用领域也不断扩大，可以设想，未来人工智能带来的科技产品，将会是人类智慧的“容器”。人工智能可以对人的意识、思维的信息过程的模拟。人工智能不是人的智能，但能像人那样思考、也可能超过人的智能。人工智能也是一门极富挑战性的科学，从事这项工作的人必须懂得计算机知识，心理学和哲学。人工智能是包括十分广泛的科学，它由不同的领域组成，如机器学习，计算机视觉等等，总的说来，人工智能研究的一个主要目标是使机器能够胜任一些通常需要人类智能才能完成的复杂工作。但不同的时代、不同的人对这种“复杂工作”的理解是不同的。

而本项目着重对智能聊天机器人进行研讨与实验，智能聊天机器人作为自然语言处理这门学科的重要产物，在人类的现代生活中相当普及。自然语言处理，即实现人机间自然语言通信，或实现自然语言理解和自然语言生成是十分困难的。造成困难的根本原因是自然语言文本和对话的各个层次上广泛存在的各种各样的歧义性或多义性。

一个中文文本从形式上看是由汉字（包括标点符号等）组成的一个字符串。由字可组成词，由词可组成词组，由词组可组成句子，进而由一些句子组成段、节、章、篇。无论在上述的各种层次：字（符）、词、词组、句子、段，……还是在下一层次向上一层次转变中都存在着歧义和多义现象，即形式上一样的一段字符串，在不同的场景或不同的语境下，可以理解成不同的词串、词组串等，并有不同的意义。一般情况下，它们中的大多数都是可以根据相应的语境和场景的规定而得到解决的。也就是说，从总体上说，并不存在歧义。这也就是我们平时并不感到自然语言歧义，和能用自然语言进行正确交流的原因。但是一方面，我们也看到，为了消解歧义，是需要极其大量的知识和进行推理的。如何将这些知识较完整地加以收集和整理出来；又如何找到合适的形式，将它们存入计算机系统中去；以及如何有效地利用它们来消除歧义，都是工作量极大且十分困难的工作。这不是少数人短时期内可以完成的，还有待长期的、系统的工作。

本项目作为自然语言处理的重要部分，先后较为详细地学习了解了以下几个方面的知识内容：同一个问题多种选择性的回答，并提供缺省回答的方案；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；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；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；基于 Rasa NLU 的本地基础聊天机器人系统的构建；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）；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；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；处理拒绝、等待状态转换和待定行动的多轮多次查询

技术等。而这些知识技能，正是聊天机器人的实现中所必不可少的，并在国内外各大应用广泛的机器人中均有所体现。

二、 研究过程及成果

（1） 同一个问题多种选择性的回答，并提供缺省回答的方案

例如当机器人被问及"你叫什么名字?"的时候，机器人可以做出"我的名字叫回音机器人","他们都叫我回音机器人","名字叫机器人，回音机器人"这样的三种回答，这就是为同一个问题提供的多种选择性回答的方案，然而，其中缺省回答的方案是第三种"名字叫机器人，回音机器人"，于是机器人便会提供这样一种缺省回答的方案。具体内容如下所示：

```
In [1]: responses = {
...         "你叫什么名字?": [
...         "我的名字叫回音机器人",
...         "他们都叫我回音机器人",
...         "名字叫机器人，回音机器人"
...         ]
...     }

In [2]: import random
...
In [3]: def respond(message):
...         if message in responses:
...             return random.choice(responses[message])
...
In [4]: respond("你叫什么名字?")
Out[4]: "名字叫机器人，回音机器人"
```

（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

正则表达式描述了一种字符串匹配的模式，可以用来检查一个串是否含有某种子串、将匹配的子串替换或者从某个串中取出符合某个条件的子串等。

例如：

`runoo+b`，可以匹配 runoob、runoooob、runooooooob 等，+ 号代表前面的字符必须至少出现一次（1次或多次）。

`runoo*b`，可以匹配 runob、runoob、runooooooob 等，* 号代表前面的字符可以不出现，也可以出现一次或者多次（0次、或1次、或多次）。

`colou?r` 可以匹配 color 或者 colour，? 问号代表前面的字符最多只可以出现一次（0次、或1次）。

构造正则表达式的方法和创建数学表达式的方法一样。也就是用多种元字符与运算符可以将小的表达式结合在一起来创建更大的表达式。正则表达式的组件可以是单个的字符、字符集合、字符范围、字符间的选择或者所有这些组件的任意组合。

正则表达式是由普通字符（例如字符 a 到 z）以及特殊字符（称为"元字符"）组成的文字模式。模式描述在搜索文本时要匹配的一个或多个字符串。正则表达式作为一个模板，将某个字符模式与所搜索的字符串进行匹配。

正则表达式在聊天机器人中的应用主要在于：判断给定字符串是否符合特定形式、抽取出字符串里的关键字符，即符合正则表达式的字符、把特定字符串转换成别的字符串等方面。

模式匹配主要指在一句话中是否可以找到的一个模式（如词组），多使用 re.research(pattern,message)来进行。

例如：

```
In [1]: import re

In [2]: pattern = "do you remember .*"

In [3]: message = "do you remember when you last watched the movie?"

In [4]: match = re.search(pattern, message)

In [5]: if match:
   :        print("string matches!")
Out[5]: string matches!
```

如上即寻找在“do you remember last watched the movie?”里面是否含有模式“do you remember”，并用 re.research(pattern,message)来进行检查，如 match，输出“string matches!”。

关键词提取指的是提取出一个句子里面的重要成分，来更好地进行判断或者回答。

例如：

```
In [1]: import re

In [2]: pattern = "if(.*)"

In [3]: message = "what would happen if the German team lost in the World Cup?"

In [4]: match = re.search(pattern, message)

In [5]: match.group(0)
Out [5]: 'if the German team lost in the World Cup?'

In [6]: match.group(1)
Out[6]: 'the German team lost in the World Cup'
```

如上即在 message 里面提取了“if the German team lost in the World Cap”和“the German team lost in the World Cap”。

句法转换主要涉及人称等的替换，I 换成 you，my 换成 your 等。如下：

```
import re
def swap_pronouns(phase):
    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
```

```
swap_pronouns("I walk my dog")
```

（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

识别意图和实体的正则表达式的优点有比机器学习方法更简单，计算效率高，但是调试正则表达式可能变得较为困难。

正则表达式的使用：

```
In [1]: re.search(r"(hello|hey|hi)", "hey there!") is not None
Out[1]: True
In [2]: re.search(r"(hello|hey|hi)", "which one?") is not None
Out[2]: True
In [3]: re.search(r"\b(hello|hey|hi)\b", "hey there!") is not None
Out[3]: True
In [4]: re.search(r"\b(hello|hey|hi)\b", "which one?") is not None
Out[4]: False
```

用正则表达式识别实体：

```
In [1]: pattern = re.compile('[A-Z]{1}[a-z]*')
In [2]: message = """
        Mary is a friend of mine,
        she studied at Oxford and
        now works at Google"""
In [3]: pattern.findall(message)
Out[3]: ['Mary', 'Oxford', 'Google']
```

如上找到在 message 里面找到了三个匹配 pattern 的单词并输出。

最近邻分类方法：需要训练数据，每个句子都被标上了他们的意图；最简单的方法：寻找最相似的标签示例，使用其意图作为最佳猜测

scikit-learn 中的最近邻分类：

```
In [1]: from sklearn.metrics.pairwise import cosine_similarity
In [2]: test_message = """
i would like to find a flight from charlotte
to las vegas that makes a stop in st. louis"""
In [3]: test_x = nlp(test_message).vector
In [4]: scores = [
   ...:     cosine_similarity(X[i,:], test_x)
   ...:    for i in range(len(sentences_train)
   ...: ]
In [5]: labels_train[np.argmax(scores)]
Out[5]: 'atis_flight'
```

支持向量机：SVM / SVC，支持向量机/分类器，如下：

```
In [1]: from sklearn.svm import SVC
In [2]: clf = SVC()
In [3]: clf.fit(X_train, y_train)
In [4]: y_pred = clf.predict(X_test)
```

（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

预建的命名实体识别（例）：

```
In [1]: import spacy
In [2]: nlp = spacy.load('en')
In [3]: doc = nlp("my friend Mary has worked at Google since 2009")
In [4]: for ent in doc.ents:
   ...:     print(ent.text, ent.label_)
   ...:

Mary PERSON
Google ORG
2009 DATE
```

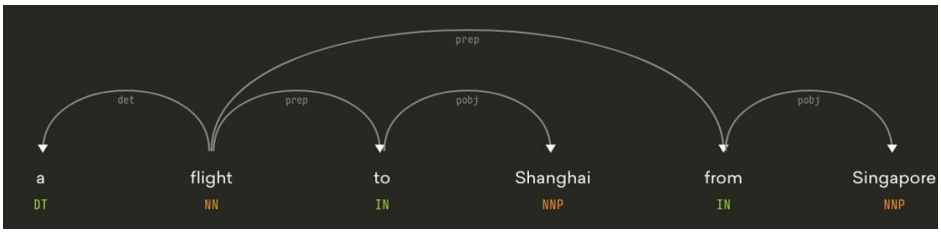
角色关系（例）：

I want a flight from Tel Aviv to Bucharest

show me flights to Shanghai from Singapore

```
In [1]: pattern_1 = re.compile('.* from (.*) to (.*)')
In [2]: pattern_2 = re.compile('.* to (.*) from (.*)')
```

依赖分析（例）：




```
In [1]: doc = nlp( 'a flight to Shanghai from Singapore')
In [2]: sh, sg = doc[3], doc[5]

In [3]: list(sh.ancestors)
Out[3]: [to, flight]

In [4]: list(sg.ancestors)
Out[4]: [from, flight]
```

（5） 基于 Rasa NLU 的本地基础聊天机器人系统的构建

Rasa NLU：用于意图识别和实体提取的库；基于 spaCy，scikit-learn 和其他库；内置支持聊天机器人特定任务。

Rasa 训练数据的格式：

```
In [1]: from rasa_nlu.training_data import load_data
In [2]: training_data = load_data(" /PATH/TO/training_data.json")
In [3]: import json
In [4]: print(json.dumps(training_data.entity_examples[0].data,
indent=2))
Out[4]: {
  "intent": "restaurant_search",
  "entities": [
    {
      "start": 31,
      "end": 36,
      "value": "north",
      "entity": "location"
    }
  ]
}
```

Interpreters：

```
In [1]: message = "I want to book a flight to London"
In [2]: interpreter.parse(message))
Out[2]: {
  "intent": {
    "name": "flight_search",
    "confidence": 0.9
```

```
  },
  "entities": [
    {
      "entity": "location",
      "value": "London",
      "start": 27,
      "end": 33
    }
  ]
}
```

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
# Create a trainer
trainer = Trainer(config.load("/PATH/TO/CONFIG_FILE"))
# Load the training data
training_data = load_data('/PATH/TO/TRAINING_DATA')
# Create an interpreter by training the model
interpreter = trainer.train(training_data)
```

（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）

数据库查询：

基本 SQL 语句：

```
SELECT * from restaurants;
SELECT name, stars from restaurants;
SELECT name from restaurants WHERE location = 'center' AND price = 'hi';
```

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

SQLite 与 Python：

```
In [1]: import sqlite3
In [2]: conn = sqlite3.connect('hotels.db')
In [3]: c = conn.cursor()
```

```
In [4]: c.execute("SELECT * FROM hotels WHERE location='south'
and price='hi' ")
Out[4]: <sqlite3.Cursor at 0x10cd5a960>

In [5]: c.fetchall()
Out[5]: [('Grand Hotel', 'hi', 'south', 5)]
```

SQL 注入：

```
# 优化前
location = “south”
price = “hi”
query = “SELECT * from restaurants where
location='{}'".format(location)
c.execute(query)
# 优化后
t = (location, price)
c.execute('SELECT * FROM hotels WHERE location=? and price=?', t)
```

使用自然语言探索数据库：

提取参数方法：

```
In [1]: message = "a cheap hotel in the north"
In [2]: data = interpreter.parse(message)
In [3]: data
Out[3]:
{
   'entities' : [
         {'end': '7' , 'entity': 'price' , 'start': 2, 'value':
'lo' },
         {'end': 26, 'entity': 'location', 'start': 21, 'value':
'north' }
      ],
   'intent': {'confidence': 0.9, 'name': 'hotel_search' }
}

In [4]: params = {}
In [5]: for ent in data["entities"]:
...:   params[ent["entity"]] = ent["value"]

In [6]: params
Out[6]: {'location': 'north', 'price': 'lo'}
```

从参数创建查询：

```
In [7]: query = “select * FROM hotels"
In [8]: filters = ["{}=?".format(k) for k in params.keys()]
In [9]: filters
Out[9]: ['price=?', 'location=?' ]
```

```
In [10]: conditions = " and ".join(filters)
In [11]: conditions
Out[11]: 'price=? and location=?'

In [12]: final_q = " WHERE ".join([query, conditions])
In [13]: final_q
Out[13]:  'SELECT * FROM hotels WHERE price=? and location=?'
```

响应：

```
In [1]: responses = [
"I'm sorry :( I couldn't find anything like that",
"what about {}?",
"{} is one option, but I know others too :)"
]
In [2]: results = c.fetchall()
In [3]: len(results)
Out[3]: 4

In [4]: index = min(len(results), len(responses)-1) 2
In [5]: responses[index]
Out[5]: '{} is one option, but I know others too :)'
```

（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

简单的记忆方法-params 参数：

```
In [1]: def respond(message, params):
            # 更新与消息中的实体的参数
            # 运行查询
            # 挑选回应
            return response, params
# 初始化参数
In [2]: params = {}
# 消息传进来
In [3]: response, params = respond(message, params)
```

否定实体技术：

例子：

no I don't want sushi

not sushi, maybe pizza?

I want burritos not sushi

假设在实体之前有“not”或“n't”意味着用户想要排除这个。绿色：正常实体，紫色：否定实体。

甄别否定实体的方法：

```
doc = nlp('not sushi, maybe pizza?')

indices = [1, 4]
ents, negated_ents = [], []

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
```

（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

实现状态机：

```
INIT = 0
CHOOSE_COFFEE = 1
ORDERED = 2
```

示例规则

```
policy_rules = {
(INIT, "order"): (CHOOSE_COFFEE, "ok, Columbian or Kenyan?"),
(CHOOSE_COFFEE, "specify_coffee"):(ORDERED, "perfect, the beans are on their way!")
}
```

使用状态机的方法：

```
In [1]: state = INIT

In [2]: def respond(state, message):
            (new_state, response) = policy_rules[(state, interpret(message))]
            return new_state, response

In [3]: def send_message(state, message):
   ...:         new_state, response = respond(state, message)
   ...:         return new_state

In [4]: state = send_message(state, message)
```

状态机使得多轮多次查询更加方便快捷。

（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

可重复使用的模式举例：
“我想要一瓶可乐”
“我很抱歉，可乐没有了。雪碧吗？”
“没问题”
Or
“我可以买一箱 200 个棕色过滤器吗”
“我很抱歉，棕色的没有了，不过我能买到白色。要为你点这些吗？“
“可以”

待处理的操作：
策略返回两个值：selected_action 和 pending_action 。
pending_action 保存在外部作用域中。
如果我们得到“是”意图且有待处理的操作，我们执行它。
如果我们得到“否”意图，我们擦除任何待处理的操作。
等待状态转换：
USER : "I'd like to order some coffee"
state = INIT
action = "request_auth" pending_state = AUTHED

BOT : Sounds good! I'd love to help you but you'll have to log in first, what's your phone number?
USER : "555-12345"
state = AUTHED
action = "acknowledge_auth" pending_state = None
Perfect! welcome back :)

三、 学习感受与收获

在参加这次远程科研项目之前，我仅仅是对人工智能、自然语言处理有着基本的理论知识了解与浓厚的兴趣，但并没有真正自己动手独立完成一个如此规模的项目过，所以此次智能聊天机器人项目，使我真切地、深刻地感受到了科研的伟大魅力与刻苦钻研精神的重要，一个个日夜的辛勤研讨也使我感受到了前所未有的充实与满足。作为第一次自己独立实现的项目，虽然最终项目的展示可能略有欠缺之处，但这宝贵的经历带给了我无限的思考与挑战，并伴随着一些快乐。在课堂上，我不仅仅学到了与智能机器人有关的技术与知识，例如模式匹配、状态转换、命名实体识别等等，更是逐步熟悉了进行科学研究的方法与布置乃至严谨认真的态度，并对未来科研的道路产生了无限的憧憬。经过近一个月的辛勤奋斗，项目也告一段落，我对自己的表现还算认可，项目的目标也已基本完成，虽然项目结束了，但是我相信它给我带来的积极影响与成长作用将伴随着我今后的科研道路，并不断地引领着我、激励着我，使我勇敢并坚持地前行。