远程科研指导项目总结报告

聊天机器人项目总结

姓　　名：　彭　友

学　　校：　上海理工大学

专　　业：　计算机科学与技术

2019 年 6 月

目录

1. 项目概述

本项目通过使用名为Scrapy的爬虫框架、名为Rasa的机器对话框架和Neo4j图数据库，构造了一个中文的医疗领域问答系统，以便人们可以通过聊天的方式快速、精准地获取到指定疾病的介绍、治疗方法、用药方案等信息，系统将具有一定的理解人类语言的能力，具有可重用性，并且与知识图谱建立联系，根据问题动态地获取数据、构造答案、返回答案。

2. 背景知识

2.1 自然语言处理

自然语言处理（Natural Language Processing）是人工智能领域中的一个研究方向，它被用于处理人类的语言。语言是人类区别其他动物的本质特性。在所有生物中，只有人类才具有语言能力。人类的多种智能都与语言有着密切的关系。人类的逻辑思维以语言为形式，人类的绝大部分知识也是以语言文字的形式记载和流传下来的。因而，NLP 是人工智能的一个重要，甚至核心的部分。语言具有模糊、上下文相关和隐晦的特性，甚至有时并不准确，同时人类的语言是基于人类认知的，而在表达时，同一个意思还可以有多种不同的表达方式。所以让机器处理人类的语言是非常困难的，而自然语言处理技术就旨在解决相关的问题。

2.2 Scrapy

Scrapy 是一套基于 Twisted 的异步处理框架，是纯 Python 实现的爬虫框架，用户只需要定制开发几个模块就可以轻松实现一个爬虫，用来抓取网页中的内容、图片等。通常单次爬取行为可分如下步骤：引擎（Engine）的作用是控制事件的触发和数据在组件中的流动，引擎根据 URL 构造请求（Request）对象传递给调度器（Scheduler），调度器将请求对象进行排队，引擎需要时，调度器就把请求数据给引擎，由引擎交给下载器（Downloader）并由下载器获取对应的响应数据（Response）交还给引擎，引擎把响应数据交给爬虫（Spider）进行解析提取，提取到的数据产生出对应的项目（Item）交给管道（Item Pipeline）进行后期过滤或者存储等。

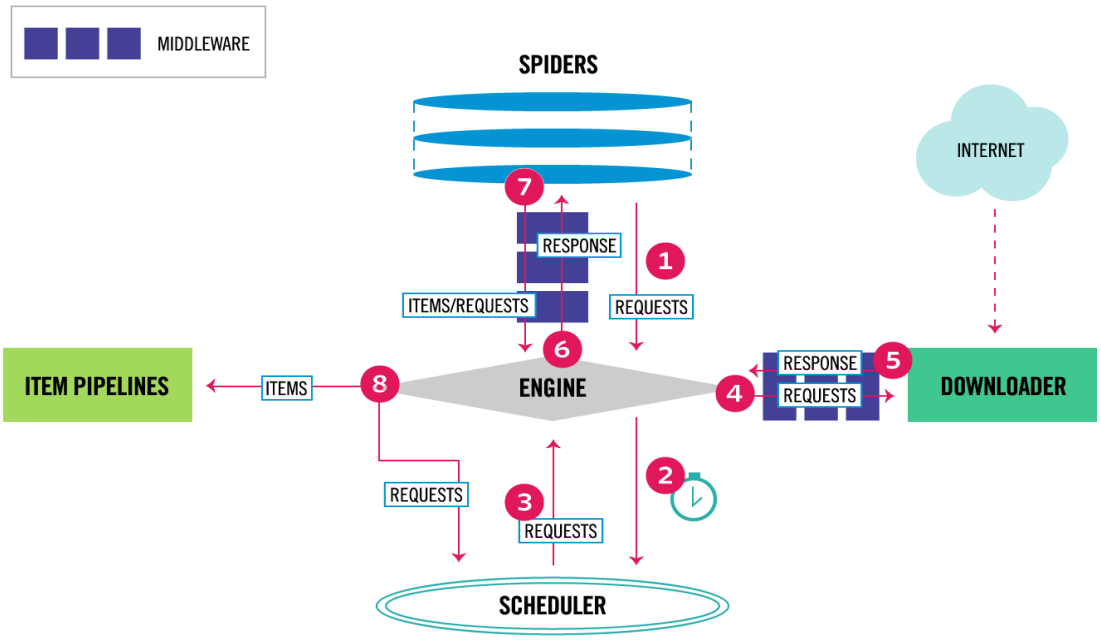

图 1 Scrapy 架构

2.3 Rasa

Rasa 是一个用于自动文本和基于语音的对话的开源机器学习框架，支持多种语言，内置支持 Spacy、MITIE、Jieba 等多种开源 NLP 工具，可将各种组件进行组合配置，构造定制化的 NLP Pipeline 以便适合需求。

Rasa 配套了 Rasa NLU 和 Rasa Core，Rasa NLU 用于实体提取和意图识别；Rasa Core 是会话引擎，它用真实的对话文本和机器学习算法训练得到模型，从而预测在收到消息后机器人该作何回复（动作）。

Rasa 有封装好的服务器 Server 程序，运行 Server 后，可以使用 Http API 来调用功能。

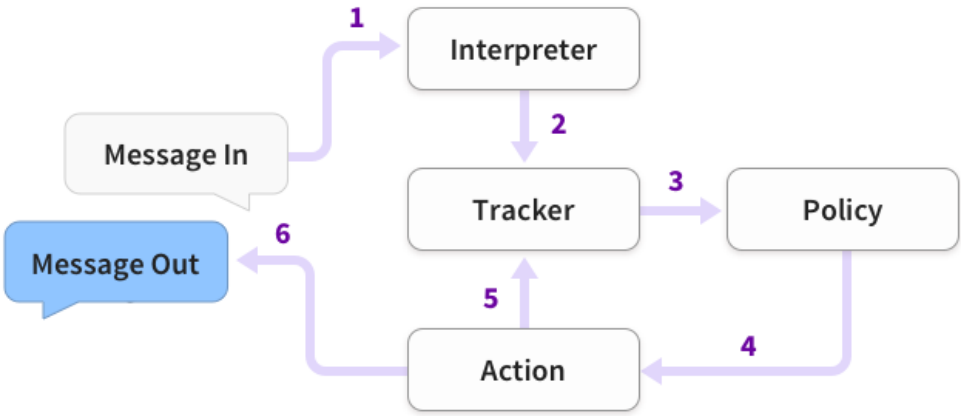

图 2 Rasa 架构机器人消息响应流程

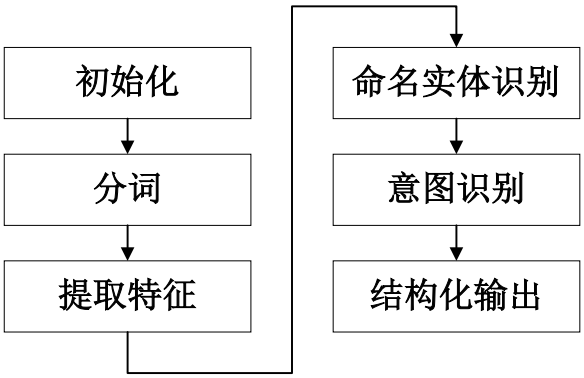

图 3 Rasa NLU Pipeline 流程

3. 过程分析

3.1 收集数据

考虑到目标是要做一个中文的医疗方面的问答系统，所以需要去搜集疾病、药物等有关的数据，但是搜寻一番后发现并没有符合需求的开放数据集，所以使用 Scrapy 库来编写爬虫程序爬取了寻医问药网。采用了随机 User-Agent、随机延迟请求和限制请求并发数等机制防止爬虫被检测，同时利用了 2 台 PC 和 1 台云服务器分别在不同网络下同时分块爬取。

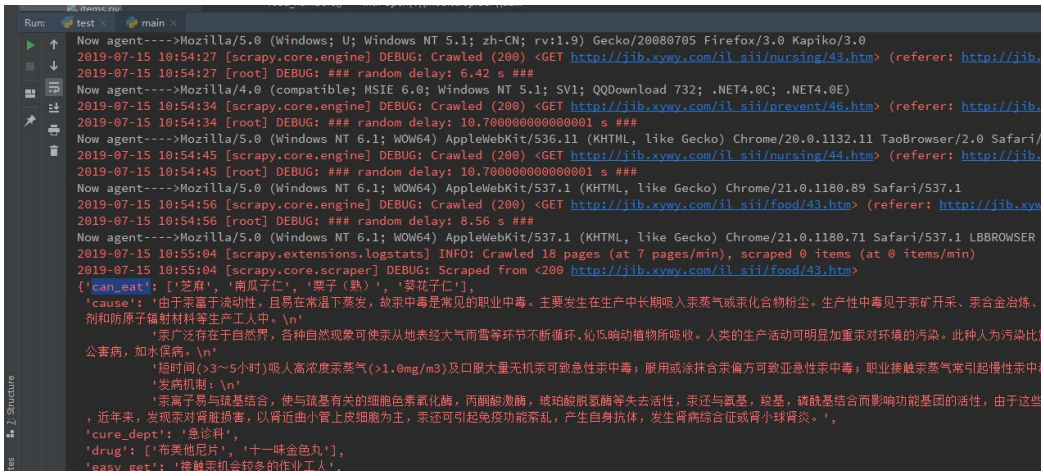

图 4 正在运行的爬虫

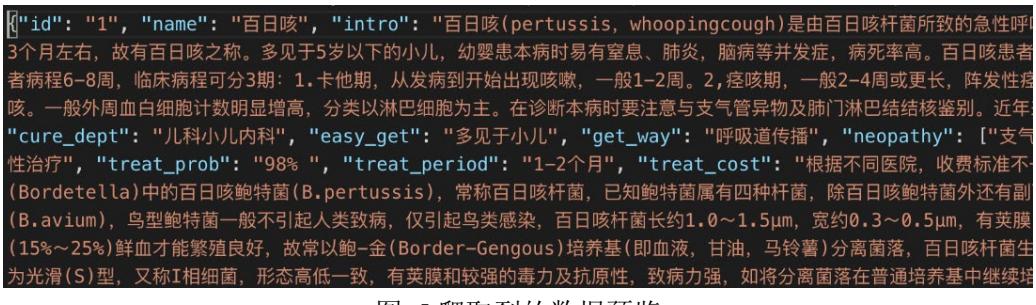

图 5 爬取到的数据预览

3.2 合并数据

由于数据是分布式爬取的，需要合并，主要通过 merge 函数实现。该函数在获取疾病字典数据后，进行去重、排序等处理，以 JSON 格式写入文件。

```
def merge(diseases: dict):
    set_dis = set()
    print('Original diseases: %d' % len(diseases))
    f = open(datadir + 'medical.json', 'a', encoding='UTF-8')
    for disease in diseases:
        disease['cure_dept'] = [dept + '科' for dept in disease['cure_dept'].split('科') if dept]
        if disease.get('treat'):
            disease['treat'] = [treat for treat in disease['treat'].split(' ') if treat]
        else:
            disease['treat'] = []
        disease = json.dumps(disease, ensure_ascii=False)
        set_dis.add(disease)
    set_dis = list(set_dis)
    sort_set_dis = sorted(set_dis, key=lambda x: int(json.loads(x)['id']))
    print('Set diseases: %d' % len(sort_set_dis))
    for disease in sort_set_dis:
        f.write(disease + '\n')
    f.close()
```

3.3 Neo4j 知识图谱设计

(1) 节点设计（Node Design）：一共设计了 5 种标签（Label）的节点。

Disease（疾病）		
属性	含义	举例
name	疾病名称	"百日咳"
intro	简介	"百日咳(pertussis，whoopingcough)是…"
cause	起因	" (一)发病原因\n 病原菌是鲍特菌属(Bordetella)…"
prevent	预防	"百日咳预防\n1、控制传染源…"
nursing	护理	"确诊的患者应立即隔离至病后 40 天…"
insurance	是否医保	"否"

easy_get	易感人群	"多见于小儿"
get_way	感染途径	"呼吸道传播"
get_prob	感染概率	"0.5%"
treat	治疗方式	"药物治疗", "支持性治疗"
treat_prob	治愈概率	"98%"
treat_period	治疗时间	"1-2 个月"
treat_cost	治疗费用	"根据不同医院，收费标准不一致，市三甲…"
treat_detail	治疗细节	"百日咳西医治疗 1、药物治疗…"
Department（科室）		
属性	含义	举例
name	科室名称	"小儿内科"
Drug（药物）		
属性	含义	举例
name	药物名称	"阿司匹林肠溶片"
Food（食物）		
属性	含义	举例
name	食物名称	"小米面"
Symptom（症状）		
属性	含义	举例
name	症状名称	"指甲呈筒形"

(2) 关系设计（Relationship Design）：共有 6 种关系。

关系类型	含义	举例
has_symptom	有症状	("大叶性肺炎", "有症状", "高热")
can_use_drug	可用药	("大叶性肺炎", "可以用药", "乳酸左氧氟沙星片")
belongs_to	属于	("大叶性肺炎", "属于", "呼吸内科")
has_neopathy	有并发症	("大叶性肺炎", "有并发症", "肺脓肿")
can_eat	可以吃	("大叶性肺炎", "可以吃", "鲫鱼")
not_eat	不能吃	("大叶性肺炎", "不能吃", "韭菜")

3.4 Rasa 的配置

(1) Rasa NLU 训练数据

要让电脑对人类的语言做出回应，首先要进行实体抽取和意图识别，Rasa NLU 正是负责这个模块，使用 Chatito 工具可以快捷地构建 Rasa NLU 所需的训练数据。

```
%[search_treat]('training': '3000', 'testing': '300')
    *[50%] ~[get?]@[disease]~[时?]~[的?]~[treat_qwd?]~[qwd?]
    *[10%] ~[treat_qwd?]@[disease]~[时?]~[的?]~[qwd?]
    *[15%] @[disease]~[是什么]~[病?]~[qwd?]
    *[15%] ~[什么是]@[disease]~[qwd?]
    *[10%] @[disease]

%[search_food]('training': '1500', 'testing': '150')
    *[30%] ~[get?]@[disease]~[时?]~[的?]~[food_qwd]~[qwd?]
    *[30%] ~[get?]@[disease]~[时?]~[的?]~[food_qwd]~[qwd?]
    *[40%] ~[get?]@[disease]~[时?]~[的?]~[可以吃?]~[什么]~[food_qwd]~[qwd?]

%[search_symptom]('training': '1500', 'testing': '150')
    *[33%] ~[get?]@[disease]~[时?]~[会?]~[不会?]~[有?]~[什么?]~[symptom_qwd]~[qwd?]
    *[33%] ~[get?]@[disease]~[的?]~[symptom_qwd]~[qwd?]
    *[33%] ~[get?]@[disease]~[时?]~[有?]~[什么?]~[symptom_qwd]~[qwd?]
```

图 6 使用 Chatito 工具制作训练数据

(2) Rasa Core 训练数据

Rasa Core 使用机器学习模型来预测机器人下一步该采取的动作，训练数据是由故事（Stories）构成，故事就是真实的对话流程，通常还会带有词槽（Slot）的填充，需要一定的人工整理。

```
## story_1_search_treat_simple
* greet
   - utter_greet
* search_treat{"disease": "百日咳"}
   - action_search_treat
* bye
   - utter_goodbye

## story_2_search_food_simple
* greet
   - utter_greet
* search_food{"disease": "百日咳"}
   - action_search_food
* bye
   - utter_goodbye
```

图 7 人工整理的对话流程

(3) Rasa 配置的选择

因为本项目是中文环境的问答系统，所以选择如下的配置：

```
pipeline:
- name: "MitieNLP"
  model:
"data/total_word_feature_extractor_zh.dat"
- name: "JiebaTokenizer"
  dictionary_path: "jieba_userdict"
- name: "MitieEntityExtractor"
- name: "EntitySynonymMapper"
- name: "RegexFeaturizer"
- name: "MitieFeaturizer"
- name: "SklearnIntentClassifier"
```

Pipeline 配置

```
policies:
  - name: "rasa.core.policies.keras_policy.KerasPolicy"
    epochs: 120
    featurizer:
      - name: MaxHistoryTrackerFeaturizer
        max_history: 5
        state_featurizer:
          - name: BinarySingleStateFeaturizer
  - name:
"rasa.core.policies.memoization.MemoizationPolicy"
    max_history: 5
  - name:
"rasa.core.policies.mapping_policy.MappingPolicy"
  - name: "rasa.core.policies.fallback.FallbackPolicy"
    nlu_threshold: 0.4
    core_threshold: 0.3
    fallback_action_name: 'action_donknow'
```

Policy 配置

(4) 现有数据的利用：

由于 Pipeline 中 MITIE 所构建的词向量维数高达 50 多万，所以训练过程对硬件要求较高，故使用了已经训练好的 data/total_word_feature_extractor_zh.dat 模型，该模型训练使用的是中文 Wikipedia Dump 和百度百科语料。

因为本项目是医疗领域，涉及大量的疾病、症状、药物等名词，同时 MITIE 的训练需要输入的是词，所以为了较好的分词效果，从已采集的数据中抽离出 Diseases、Departments、Drugs、Symptoms 和 Foods 作为 Jieba 分词组件的自定义词典。

3.5 前端

考虑到 Rasa 框架中带有封装好的可启动的服务器程序，Rasa 官方也推荐了基于 React 和 Socket 的开源聊天前端框架 Rasa-WebChat，它可以返回并渲染 Markdown 格式文本，于是本项目采用了网页的形式，并引入了 WebChat.js，通过 Http 请求调用 Rasa 服务。

图 8 点击箭头所指的图标

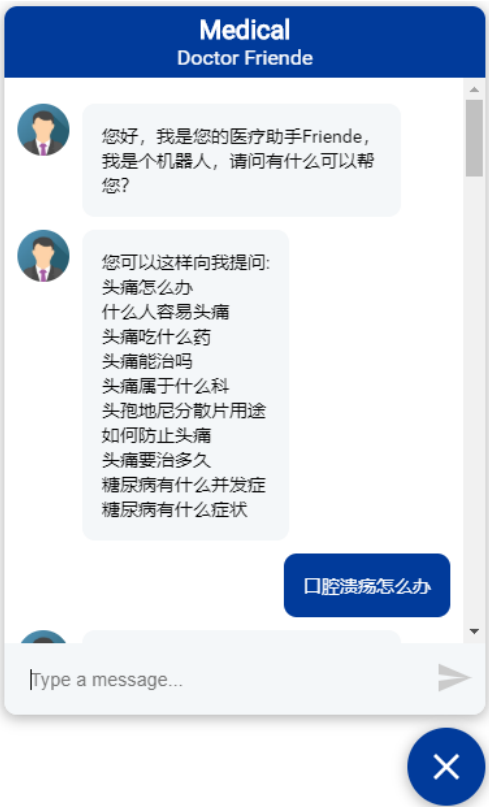

图 9 聊天框效果图

4. 结果展示

4.1 可用功能

意图	含义	示例
search_treat	查询疾病治疗方法	头痛怎么办
search_food	患某病能、不能吃的食物	头痛能吃什么
search_symptom	查询疾病症状	头痛有什么症状
search_cause	查询疾病起因	怎么会头痛
search_neopathy	查询疾病并发症	头痛的并发症
search_drug	查询疾病可用药	头痛吃什么药
search_prevention	查询疾病预防方法	头痛怎么预防
search_drug_func	查询药物能治疗的疾病	牛黄解毒丸能治啥
search_disease_treat_time	查询疾病治疗时长	头痛要治多久
search_easy_get	查询疾病易感人群	什么人容易得头痛
search_disease_dept	查询疾病所属科室	头痛属于什么科

4.2 知识图谱

图 10 某一疾病的关系结构

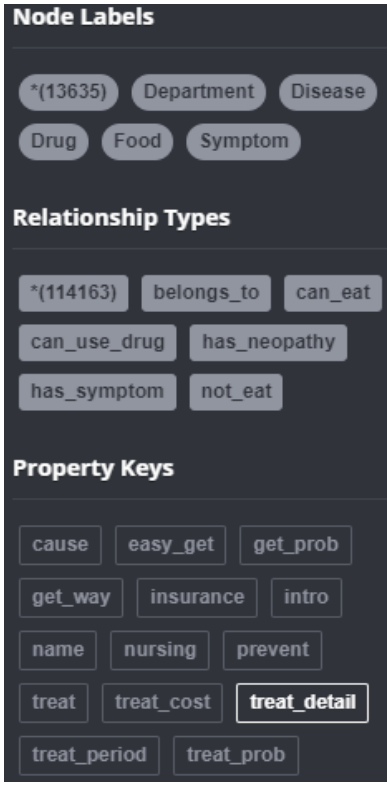

图 11 图谱概况一览

4.3 前端

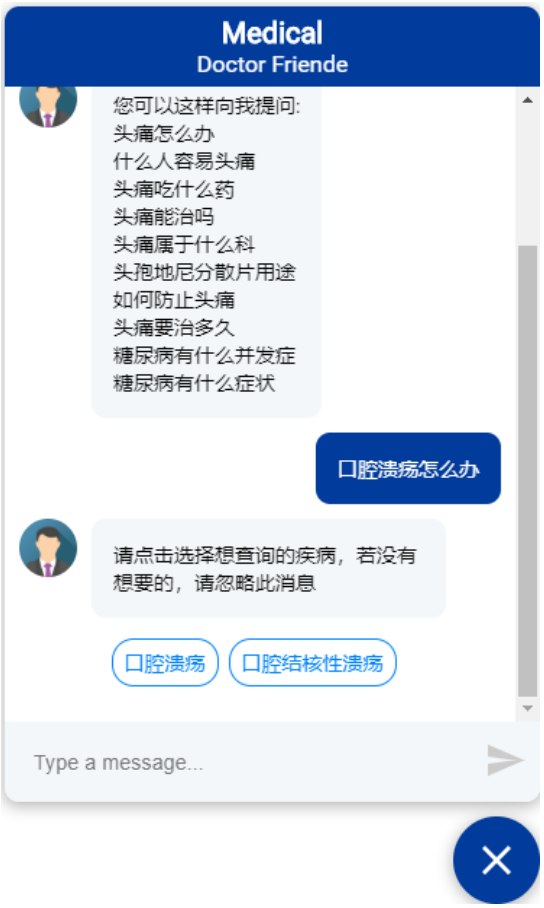

图 12 对疾病名称的模糊搜索

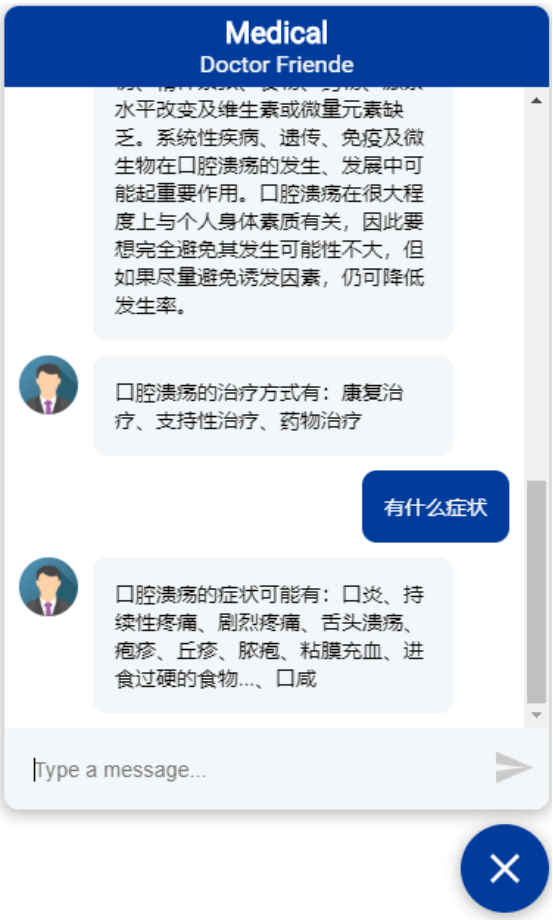

图 13 多轮查询记忆疾病名

5. 感想与收获

本次“聊天机器人”远程指导项目为期一个月，虽然“自然语言处理”的字眼已在我的学习生活中出现多次，但我一直没有实际地去学习它，同样我也不曾想过它会以“聊天机器人项目”的形式与我碰面。

在参与项目过程中，老师讲授了自然语言处理的发展和应用场景、正则表达式的应用、词向量的构成、命名实体识别、意图识别、否定、多轮查询和有状态对话等知识点，概念容易理解，但是真正使用时却困难重重。

到了项目制作的阶段，就有一种“病来如山倒病去如抽丝”的感觉。从确定医疗领域开始到最后完成一个基础的聊天机器人，我划分了如下阶段：

(1) 构想阶段：

- 搜索是否有前人做过类似的程序，并试用、学习；

(2) 数据阶段：

- 发现现有可取的数据较少，亦不想直接取用他人整理的数据，于是学

习爬虫技术，学习 Scrapy 框架；

- 寻找医疗方面的网站并尝试爬取数据；
- 整理、存储数据，并设计数据入库的结构；
- 学习 Neo4j 的使用，学习 py2neo 库；
- 熟悉 Rasa NLU 和 Rasa Core 的训练数据的结构，学习使用 Chatito 工具构造数据；

(3) 构造阶段：

- 学习 Rasa 框架的结构、工作流程；
- 学习国人制作的 rasa_nlu_chi，跑通 demo、进一步熟悉 Rasa；
- 熟悉新版 Rasa 与旧版的一些变化，并尝试用新版跑通 demo；
- 学习 Rasa 自定义 action 的用法，并设计 Rasa Core 的响应逻辑。
- 训练模型，命令行调试。
- 制作 Web 前端界面，调通程序。

(4) Debug 阶段：

- 尝试在本地聊天，并在发现问题后补充、修正数据集，重新训练，修正 action。

(5) 部署阶段：

- 将前后端部署到云服务器上，对应地修改配置文件。

过程不可谓不充实，从项目出发，发现问题，学习解决问题的技术，利用技术解决问题，形成一个循序渐进、螺旋式的学习过程。

本次远程指导项目收获的不只是知识，由于是单人完成，所以更锻炼了自己做科研时应有的条理性、逻辑性的思维方式。

6. 参考

[1] 刘焕勇 QABasedOnMedicalKnowledgeGraph

[2] 王冠 Rasa_NLU_Chi

[3] 孔晓泉 WeatherBot

[4] 孔晓泉 Rasa NLU 的 Pipeline 和 Component

[5] mrbot-ai Rasa-WebChat

[6] Rasa Official Rasa-Doc

[7] Rasa Official Legacy-Rasa-Doc

[8] Rasa Official Rasa-Forum