基于知识图谱的问答系统

姓名：张朋洲　　学号：2020104259

基于知识图谱的问答系统

目录

一、背景

1．背景分析

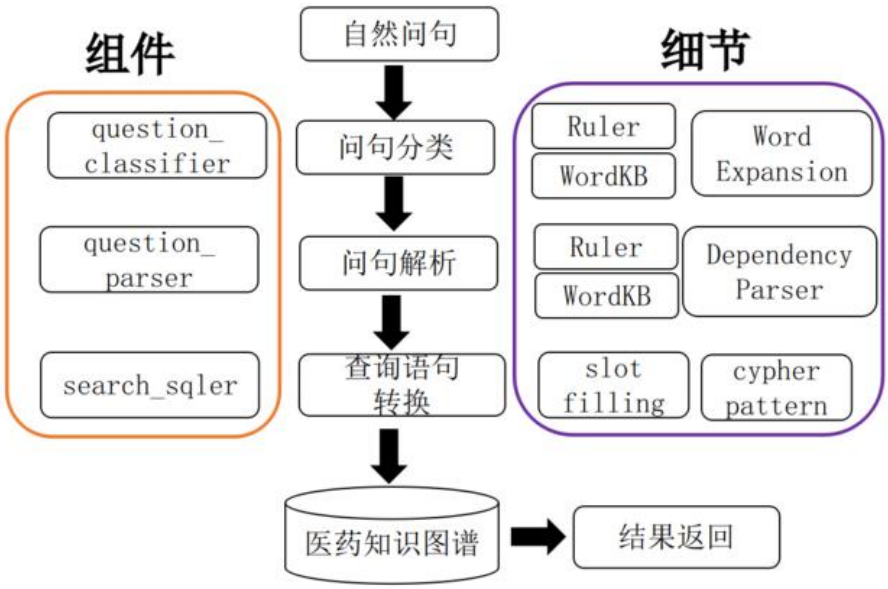
目前互联网搜索引擎技术高速发展，已经有百度，谷歌巨头公司，搜索引擎技术的基本技术思路是，用户输入查找问题，引擎锁定关键词，在数据库筛选与关键词关联较大的信息，将其排序反馈至用户端，用户在此基础上筛选有用信息。整体来看，搜索引擎技术限制明显，无法精准锁定客户需要，对于用户快速锁定答案的需求无法满足，问答系统显然有别于搜索引擎，可以精准捕捉用户的自然语言问句的精准信息，并就问题同用户完成信息的交互，实现精准问答。

2. 介绍

拟设计一个基于知识图谱的智能问答系统，并应当保证该智能问答系统可以回答 4 个及其以上的问题。由于本实验室目前正在使用知识图谱搭建问答系统，故而这里将使用知识图谱的方式构建该智能问答系统。这里将构建一个关于医药领域的问答系统。以“肺气肿”为例，本系统应当能够回答疾病的简单介绍，相关症状，推荐食谱，治疗方法，使用的药品等。

二、项目主体结构

项目主体结构如下图所示：



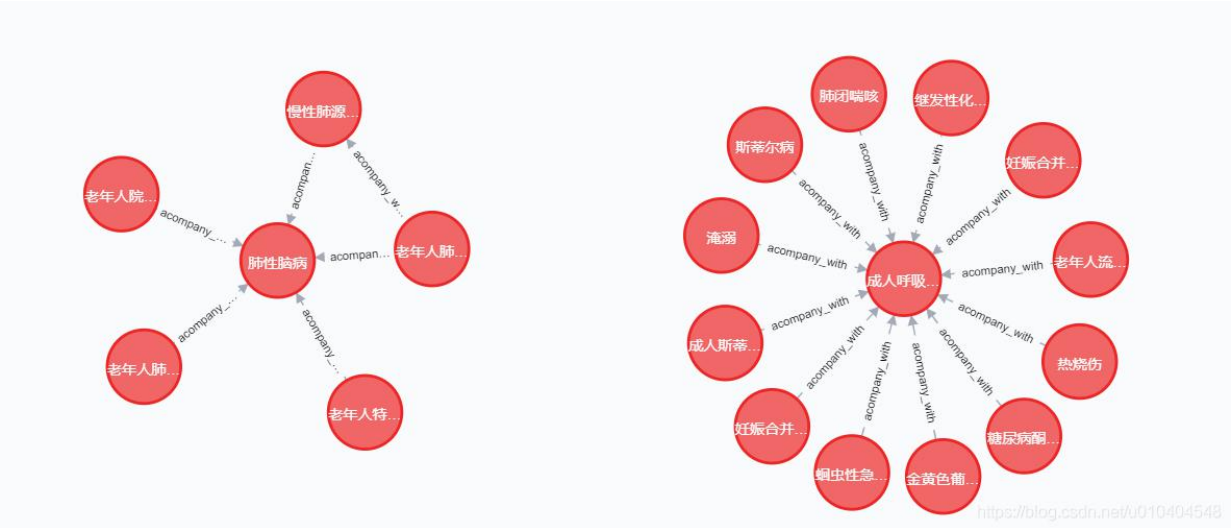
基于知识图谱的问答框架

该项目分为知识图谱的构建、基于知识图谱的问答两部分，build_medicalgraph.py 构建图谱、chatbot_graph.py 启动问题系统，项目中的文件层次结构如下：

```
├── data
│   ├── medical.json
├── dict
│   ├── check.txt
│   ├── deny.txt
│   ├── department.txt
│   ├── disease.txt
│   ├── drug.txt
│   ├── food.txt
│   ├── producer.txt
│   ├── symptom.txt
├── prepare_data
│   ├── build_data.py
│   ├── data_spider.py
│   ├── disease.txt
│   ├── first_name.txt
│   ├── first_name_spider.py
│   ├── max_cut.py
├── answer_search.py
├── build_medicalgraph.py
├── chatbot_graph.py
├── question_classifier.py
├── question_parser.py
```

三．知识图谱的构建

根据字典形式的数据创建结点，以疾病为中心定义关系形成三元组表示的知识，将结点和关系导入 neo4j 数据库形成知识图谱，见脚本 build_medicalgraph.py。下图是根据清洗后的数据生成的知识图谱。



四、问题系统构成

问题系统是该项目的核心部分，主要由问句分类、问句解析、答案搜索三部分构成。

（一）问句分类

问句分类原项目采用了“两步走”的方式，第一步是用 AC 自动机算法识别问题中的关键字，根据事先构建的“实体：类型”大型字典确认关键字的类型。第二步是基于规则的方法，依据经验构建数种疑问句关键词字典，再根据这些关键字是否存在于问句来确定问句的类型。实际上，类型的确定就是一个多分类的问题。问句分类是问答部分的核心。

问句中的关键词匹配：

```
self.symptom_qwds = ['症状', '表征', '现象', '症候', '表现']
self.cause_qwds = ['原因','成因', '为什么', '怎么会', '怎样才', '咋样才', '怎样会', '如何会', '为啥', '为何', '如何才会', '怎么才会', '会导致', '会造成']
self.acompany_qwds = ['并发症', '并发', '一起发生', '一并发生', '一起出现', '一并出现', '一同发生', '一同出现', '伴随发生', '伴随', '共现']
self.food_qwds = ['饮食', '饮用', '吃', '食', '伙食', '膳食', '喝', '菜' ,'忌口', '补品', '保健品', '食谱', '菜谱', '食用', '食物','补品']
self.drug_qwds = ['药', '药品', '用药', '胶囊', '口服液', '炎片']
self.prevent_qwds = ['预防', '防范', '抵制', '抵御', '防止','躲避','逃避','避开','免得','逃开','避开','避掉','躲开','躲掉','绕开',
                     '怎样才能不', '怎么才能不', '咋样才能不','咋才能不', '如何才能不',
                     '怎样才不', '怎么才不', '咋样才不','咋才不', '如何才不',
                     '怎样才可以不', '怎么才可以不', '咋样才可以不', '咋才可以不', '如何可以不',
                     '怎样才可不', '怎么才可不', '咋样才可不', '咋才可不', '如何可不']
self.lasttime_qwds = ['周期', '多久', '多长时间', '多少时间', '几天', '几年', '多少天', '多少小时', '几个小时', '多少年']
self.cureway_qwds = ['怎么治疗', '如何医治', '怎么医治', '怎么治', '怎么医', '如何治', '医治方式', '疗法', '咋治', '怎么办', '咋办', '咋治']
self.cureprob_qwds = ['多大概率能治好', '多大几率能治好', '治好希望大么', '几率', '几成', '比例', '可能性', '能治', '可治', '可以治', '可以医']
self.easyget_qwds = ['易感人群', '容易感染', '易发人群', '什么人', '哪些人', '感染', '染上', '得上']
self.check_qwds = ['检查', '检查项目', '查出', '检查', '测出', '试出']
self.belong_qwds = ['属于什么科', '属于', '什么科', '科室']
self.cure_qwds = ['治疗什么', '治啥', '治疗啥', '医治啥', '治愈啥', '主治啥', '主治什么', '有什么用', '有何用', '用处', '用途',
                  '有什么好处', '有什么益处', '有何益处', '用来', '用来做啥', '用来作甚', '需要', '要']
```

根据匹配到的关键词分类问句

```
# 症状
if self.check_words(self.symptom_qwds, question) and ('disease' in types
    question_type = 'disease_symptom'
    question_types.append(question_type)

if self.check_words(self.symptom_qwds, question) and ('symptom' in types
    question_type = 'symptom_disease'
    question_types.append(question_type)

# 原因
if self.check_words(self.cause_qwds, question) and ('disease' in types):
    question_type = 'disease_cause'
    question_types.append(question_type)
# 并发症
if self.check_words(self.acompany_qwds, question) and ('disease' in type
    question_type = 'disease_acompany'
    question_types.append(question_type)

# 推荐食品
if self.check_words(self.food_qwds, question) and 'disease' in types:
    deny_status = self.check_words(self.deny_words, question)
    if deny_status:
        question_type = 'disease_not_food'
    else:
        question_type = 'disease_do_food'
    question_types.append(question_type)

#已知食物找疾病
if self.check_words(self.food_qwds+self.cure_qwds, question) and 'food'
    deny_status = self.check_words(self.deny_words, question)
    if deny_status:
        question_type = 'food_not_disease'
    else:
        question_type = 'food_do_disease'
    question_types.append(question_type)

# 推荐药品
if self.check_words(self.drug_qwds, question) and 'disease' in types:
    question_type = 'disease_drug'
    question_types.append(question_type)

# 药品治啥病
if self.check_words(self.cure_qwds, question) and 'drug' in types:
    question_type = 'drug_disease'
    question_types.append(question_type)
```

（二）问句解析

问句解析的实质是根据问句类型，选择适当的neo4j的match匹配语句。

```
'''解析主函数'''
def parser_main(self, res_classify):
    args = res_classify['args']
    entity_dict = self.build_entitydict(args)
    question_types = res_classify['question_types']
    sqls = []
    for question_type in question_types:
        sql_ = {}
        sql_['question_type'] = question_type
        sql = []
        if question_type == 'disease_symptom':
            sql = self.sql_transfer(question_type, entity_dict.get('disease'))

        elif question_type == 'symptom_disease':
            sql = self.sql_transfer(question_type, entity_dict.get('symptom'))

        elif question_type == 'disease_cause':
            sql = self.sql_transfer(question_type, entity_dict.get('disease'))

        elif question_type == 'disease_acompany':
            sql = self.sql_transfer(question_type, entity_dict.get('disease'))

        elif question_type == 'disease_not_food':
            sql = self.sql_transfer(question_type, entity_dict.get('disease'))

        elif question_type == 'disease_do_food':
            sql = self.sql_transfer(question_type, entity_dict.get('disease'))

        elif question_type == 'food_not_disease':
            sql = self.sql_transfer(question_type, entity_dict.get('food'))

        elif question_type == 'food_do_disease':
            sql = self.sql_transfer(question_type, entity_dict.get('food'))

        elif question_type == 'disease_drug':
            sql = self.sql_transfer(question_type, entity_dict.get('disease'))

        elif question_type == 'drug_disease':
            sql = self.sql_transfer(question_type, entity_dict.get('drug'))

        elif question_type == 'disease_check':
            sql = self.sql_transfer(question_type, entity_dict.get('disease'))

        elif question_type == 'check_disease':
            sql = self.sql_transfer(question_type, entity_dict.get('check'))
```

查找相关数据：

```
# 查询语句
sql = []
# 查询疾病的原因
if question_type == 'disease_cause':
    sql = ["MATCH (m:Disease) where m.name = '{0}' return m.name, m.cause".format

# 查询疾病的防御措施
elif question_type == 'disease_prevent':
    sql = ["MATCH (m:Disease) where m.name = '{0}' return m.name, m.prevent".forma

# 查询疾病的持续时间
elif question_type == 'disease_lasttime':
    sql = ["MATCH (m:Disease) where m.name = '{0}' return m.name, m.cure_lasttime'

# 查询疾病的治愈概率
elif question_type == 'disease_cureprob':
    sql = ["MATCH (m:Disease) where m.name = '{0}' return m.name, m.cured_prob".fo

# 查询疾病的治疗方式
elif question_type == 'disease_cureway':
    sql = ["MATCH (m:Disease) where m.name = '{0}' return m.name, m.cure_way".forr

# 查询疾病的易发人群
elif question_type == 'disease_easyget':
    sql = ["MATCH (m:Disease) where m.name = '{0}' return m.name, m.easy_get".forr

# 查询疾病的相关介绍
elif question_type == 'disease_desc':
    sql = ["MATCH (m:Disease) where m.name = '{0}' return m.name, m.desc".format(

# 查询疾病有哪些症状
elif question_type == 'disease_symptom':
    sql = ["MATCH (m:Disease)-[r:has_symptom]->(n:Symptom) where m.name = '{0}' re

# 查询症状会导致哪些疾病
elif question_type == 'symptom_disease':
    sql = ["MATCH (m:Disease)-[r:has_symptom]->(n:Symptom) where n.name = '{0}' re

# 查询疾病的并发症
elif question_type == 'disease_acompany':
    sql1 = ["MATCH (m:Disease)-[r:acompany_with]->(n:Disease) where m.name = '{0}'
    sql2 = ["MATCH (m:Disease)-[r:acompany_with]->(n:Disease) where n.name = '{0}'
    sql = sql1 + sql2
# 查询疾病的忌口
elif question_type == 'disease_not_food':
    sql = ["MATCH (m:Disease)-[r:no_eat]->(n:Food) where m.name = '{0}' return m.n
```

（三）答案搜索

访问neo4j数据库，执行问句解析得到了match语句，并将执行结果转化为用户可以接受的形式

```
    return ''
if question_type == 'disease_symptom':
    desc = [i['n.name'] for i in answers]
    subject = answers[0]['m.name']
    final_answer = '{0}的症状包括：{1}'.format(subject, '；'.join(list(set(desc))[:

elif question_type == 'symptom_disease':
    desc = [i['m.name'] for i in answers]
    subject = answers[0]['n.name']
    final_answer = '症状{0}可能染上的疾病有：{1}'.format(subject, '；'.join(list(set(

elif question_type == 'disease_cause':
    desc = [i['m.cause'] for i in answers]
    subject = answers[0]['m.name']
    final_answer = '{0}可能的成因有：{1}'.format(subject, '；'.join(list(set(desc))

elif question_type == 'disease_prevent':
    desc = [i['m.prevent'] for i in answers]
    subject = answers[0]['m.name']
    final_answer = '{0}的预防措施包括：{1}'.format(subject, '；'.join(list(set(desc)

elif question_type == 'disease_lasttime':
    desc = [i['m.cure_lasttime'] for i in answers]
    subject = answers[0]['m.name']
    final_answer = '{0}治疗可能持续的周期为：{1}'.format(subject, '；'.join(list(set(

elif question_type == 'disease_cureway':
    desc = [';'.join(i['m.cure_way']) for i in answers]
    subject = answers[0]['m.name']
    final_answer = '{0}可以尝试如下治疗：{1}'.format(subject, '；'.join(list(set(des

elif question_type == 'disease_cureprob':
    desc = [i['m.cured_prob'] for i in answers]
    subject = answers[0]['m.name']
    final_answer = '{0}治愈的概率为（仅供参考）：{1}'.format(subject, '；'.join(list(

elif question_type == 'disease_easyget':
    desc = [i['m.easy_get'] for i in answers]
    subject = answers[0]['m.name']

    final_answer = '{0}的易感人群包括：{1}'.format(subject, '；'.join(list(set(desc)

elif question_type == 'disease_desc':
    desc = [i['m.desc'] for i in answers]
    subject = answers[0]['m.name']
    final_answer = '{0},熟悉一下：{1}'.format(subject,  '；'.join(list(set(desc))[:
```

五、实验结果：

六、项目总结

基于规则的问答系统没有复杂的算法，一般采用模板匹配的方式寻找匹配度最高的答案，回答结果依赖于问句类型、模板语料库的覆盖全面性，面对已知的问题，可以给出合适的答案，对于模板匹配不到的问题或问句类型，经常遇到的有三种回答方式：

1、给出一个无厘头的答案；

2、婉转的回答不知道，提示用户换种方式去问；

3、转移话题，回避问题；

例如，本项目中采用了婉转的方式回答不知道