基于知识图谱的医疗问答系统设计报告

姓名	学号	贡献率
郭鉴豪	12021224	25%
朱赟	12021051	25%
李冰斌	22021248	25%
李洋	22021251	25%

基于知识图谱的医疗问答系统设计报告
1 项目简介
1.1 开发环境、开发工具及运行要求
2 技术细节
2.1 知识图谱背景
2.2 医疗知识图谱构建
2.3 自动问答架构
3 操作说明

1 项目简介

1.1 开发环境、开发工具及运行要求

- 开发环境：
  - 操作系统：Windows10 教育版
  - CPU：Intel(R) Core(TM) i5-6300HQ CPU @ 2.30GHz
  - 内存：8.00GB RAM
  - 显卡：NVIDIA GeForce GTX 950M
  - IDE：PyCharm 2017.1
- 开发工具
  - neo4j-community-3.5.5
  - pygame 1.9.4
  - py2neo 4.3.0
  - tkinter
- 运行要求
  - 安装neo4j数据库管理系统；
  - 安装py2neo，用python可以连接neo4j；
  - 安装tkinter和pygame前端开发工具；

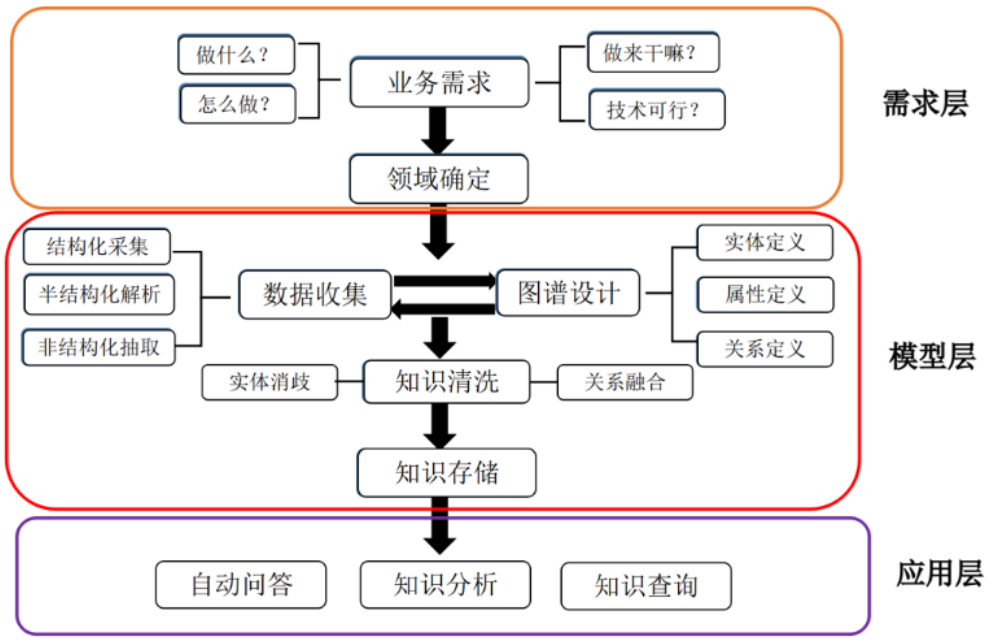
2 技术细节

2.1 知识图谱背景

- 知识图谱是通过将应用数学、图形学、信息可视化技术、信息科学等学科的理论与方法与计量学引文分析、共现分析等方法结合，并利用可视化的图谱形象地展示学科的核心结构、发展历史、前沿领域以及整体知识架构达到多学科融合目的的现代理论。它把复杂的知识领域通过数据挖掘、信息处理、知识计量和图形绘制而显示出来，揭示知识领域的动态发展规律，为学科研究提供切实的、有价值的参考；
- 问答系统也是知识图谱的典型应用场景。目前在基于知识图谱的问答系统中采用的方法主要包括：基于信息提取的方法，利用问句信息结合知识库资源获取候选答案；基于语义解析的方法，将自然语言问句解析成一种逻辑表达形式，通过这种结构化表达从知识库中寻找答案；基于向量空间建模的方法，使用向量空间描述自然语言问句以及知识图谱中的实体和关系，通过机器学习、深度学习等方法生成问答模型进行回答；
- 知识图谱的前身是语义网，它吸收了语义网、本体在知识组织和表达方面的理念，使得知识更易于在计算机之间和计算机与人之间交换、流通和加工。具体来说，一个知识图谱由模式图、数据图及两者之间的关系组成：模式图对人类知识领域的概念层面进行描述，强调概念及概念关系的形式化表达，模 式图中节点是概念实体，边是概念间的语义关系，如 part-of；数据图对物理世界层面进行描述，强调一系列客观事实。数据图中的节点有两类，一是模式图中的概念实体，二是描述性字符串，数据图中的边是具体事实的语义描述；模式图和数据图之间的关系指数据图的实例与模式图的概念之间的对应，或者说模式图是数据图的模具；
- 近年来，在人工智能的蓬勃发展下，知识图谱涉及到的知识抽取、表示、融合、推理、问答等关键问题得到一定程度的解决和突破，知识图谱成为知识服务领域的一个新热点，受到国内外学者和工业界广泛关注。，它以独有的技术优势顺应了信息化时代的发展，比如渐增式的数据模式设计；良好的数据集成；现有 RDF、OWL等标准支持；语义搜索和知识推理能力等。在医学领域，随着区域卫生信息化及医疗信息系统的发展，积累了海量的医学数据。如何从这些数据中提炼信息，并加以管理、共享及应用，是推进医学智能化的关键问题，是医学知识检索、临床诊断、医疗质量管理、电子病历 及健康档案智能化处理的基础；

2.2 医疗知识图谱构建

- 知识图谱构建框架
  首先我们需要构建知识图谱。整个系统分成三层，首先是需求层，我们这里的需求是构建一个基于知识图谱的自动问答系统，可以对医药治疗领域的相关问题进行自动问答。然后是模型层，模型层这一部分需要收集数据并导入数据库，数据包括结构化数据和半结构化数据，导入数据后需要进行知识图谱的设计，定义相关的实体，属性和关系。完成这两层的设计之后就可以用于应用层来进行自动问答了。图模型如下：



- 知识图谱实体类型

实体类型	中文含义
Check	诊断检查项目
Department	医疗科目
Disease	疾病
Drug	药品
Food	食物
Producer	在售药品
Symptom	疾病症状

- 知识图谱实体关系类型

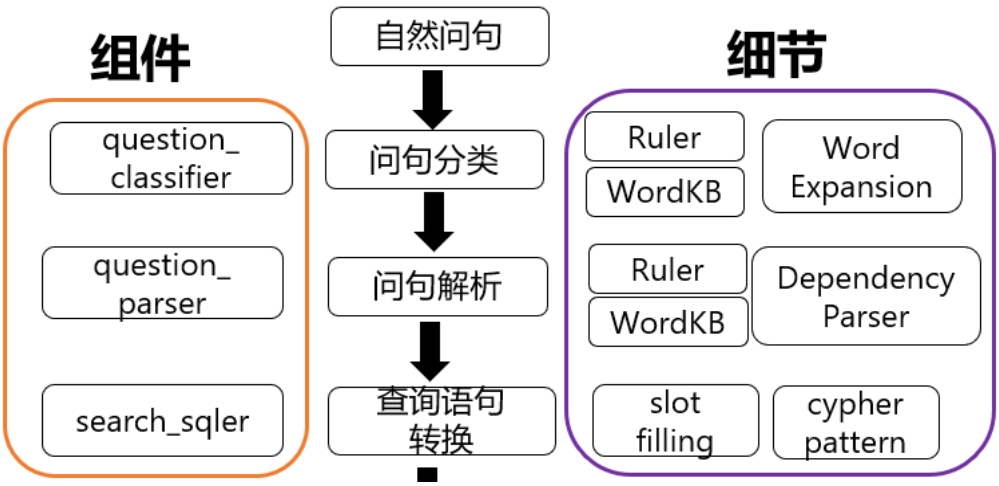
实体关系类型	中文含义
belongs_to	属于
common_drug	疾病常用药品
do_eat	疾病宜吃食物
drugs_of	药品在售药品
need_check	疾病所需检查
no_eat	疾病忌吃食物
recommand_drug	疾病推荐药品
recommand_eat	疾病推荐食谱
has_symptom	疾病症状
acompany_with	疾病并发疾病

- 知识图谱属性类型

属性类型	中文含义
name	疾病名称
desc	疾病简介
cause	疾病病因
prevent	预防措施
cure_lasttime	治疗周期
cure_way	治疗方式
cured_prob	治愈概率
easy_get	疾病易感人群

2.3 自动问答架构

完成构建知识图谱之后，就可以利用知识图谱来进行自动问答。整个自动问答的架构如下图所示，系统首先会对自然问句进行分类，通过classifier进行问句的分类，分类之后会对问句进行解析，转化为数据库的查询语句并返回结果。其中图模型如下：



- 问句分类 我们通过实现QuestionClassifier这个类来进行问句的分类，分类的方法很简单，主要是通过检测问句中的特征词来进行的，然后根据相应的特征词来把问句分成所需的类型。
    - 基于特征词进行分类

```
'''基于特征词进行分类'''
def check_words(self, wds, sent):
    for wd in wds:
        if wd in sent:
            return True
    return False
```

- 分类函数 当对问题进行分类的时候，我们需要收集问句中所涉及到的实体类型，区分是哪种实体。

```
#收集问句当中所涉及到的实体类型
types = []
for type_ in medical_dict.values():
    types += type_
question_type = 'others'

question_types = []

# 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)

# 若没有查到相关的外部查询信息，那么则将该疾病的描述信息返回
if question_types == [] and 'disease' in types:
    question_types = ['disease_desc']

# 若没有查到相关的外部查询信息，那么则将该疾病的描述信息返回
if question_types == [] and 'symptom' in types:
    question_types = ['symptom_disease']

# 将多个分类结果进行合并处理，组装成一个字典
data['question_types'] = question_types
return data
```

- 问题解析
  - 通过构建一个QuestionPaser的类，我们将以分类的问题进行解析，首先需要构建实体节点。

```
'''构建实体节点'''
def build_entitydict(self, args):
    entity_dict = {}
    for arg, types in args.items():
        for type in types:
            if type not in entity_dict:
                entity_dict[type] = [arg]
            else:
                entity_dict[type].append(arg)

    return entity_dict
```

- 解析问题的主函数部分，通过判断查询问题的类型，我们需要转换成对应的查询语句。

```
'''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'))
    if sql:
        sql_['sql'] = sql
            sqls.append(sql_)
    return sqls
```

- 针对不同的问题，我们需要分开进行处理。

```
'''针对不同的问题，分开进行处理'''
def sql_transfer(self, question_type, entities):
    if not entities:
        return []

    # 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(i) for i in entities]
            # 查询疾病应该进行的检查
    elif question_type == 'disease_check':
        sql = ["MATCH (m:Disease)-[r:need_check]->(n:Check) where m.name = '{0}' return
m.name, r.name, n.name".format(i) for i in entities]

    # 已知检查查询疾病
    elif question_type == 'check_disease':
```

```
        sql = ["MATCH (m:Disease)-[r:need_check]->(n:Check) where n.name = '{0}' return m.name, r.name, n.name".format(i) for i in entities]
    return sql
```

- 问答系统运行，这里定义了一个叫ChatBotGraph的类。
  - 首先对ChatBotGraph进行初始化。

```
def __init__(self):
    self.classifier = QuestionClassifier()
    self.parser = QuestionPaser()
    self.searcher = AnswerSearcher()
```

  - 问答系统的主函数。

```
def chat_main(self, sent):
    res_classify = self.classifier.classify(sent)
    if not res_classify:
        return answer
    res_sql = self.parser.parser_main(res_classify)
    final_answers = self.searcher.search_main(res_sql)
    if not final_answers:
        return answer
    else:
        return '\n'.join(final_answers)
```

3 操作说明

- 在官网下载安装包并安装neo4j数据库管理系统；
- 配置neo4j的cmd运行环境（在系统环境变量中添加neo4j安装目录下的bin目录）；
- 在管理员模式下运行cmd，输入neo4j.bat console运行neo4j；
- 等neo4j运行完毕，在浏览器中打开http://localhost:7474/，并修改neo4j的密码（初始账户密码均为neo4j，需要登录修改一次才能正常使用）；
- 修改build_medicalgraph.py以及answer_search.py中关于neo4j连接的信息（帐号和密码）
- 运行build_medicalgraph.py，将数据导入neo4j中建立知识图谱（进入源代码目录下，python build_medicalgraph.py）；
- 运行chat_ui.py，打开主程序即可（进入源代码目录下，python chat_ui.py）；
- 主界面分为两个文本框，其中下面的框为输入。在下面的框中输入信息后，点击右下角上方的按钮即可发送信息并得到回答（或者点击键盘的<Up>键位也可得到相同结果）；
- 主界面右下角下方的按钮用于tts发音（注意：Windows系统下不会自己生成目录文件，所以需要在源代码所在目录下创建output_file目录用于存放语音文件）