# 第6章 基于知识图谱的电影问答系统

通过上一章节的学习，我们已经了解了基于知识库的问答系统的的相关内容以及如何基于Neo4j数据库构建知识图谱数据。本章节我们将利用上一个章节构建的电影知识图谱搭建一个关于电影知识的问答系统，你不需要使用同样的例子，随意选择一个案例，按照本章的步骤进行操作，在本章结束的时候即可完成一个问答系统的搭建。

下面将开始搭建一个基于知识图谱的电影问答系统，它能够理解用户的查询意图，并将用户查询的相关电影知识返回给用户。

本章主要涉及到的知识点有：

* 系统介绍：介绍系统实现的框架。
* 基于知识图谱的电影知识问答系统实现：介绍项目中用到的数据集的采集整理。

注意：本章代码地址：xxx

## 6.1 系统介绍

本节介绍一下基于知识图谱的电影知识问答系统的主要功能及系统实现的框架。

### 6.1.1 基于知识图谱的电影问答系统

首先要确定一下基于知识图谱的电影问答系统能做什么，我们的问答系统是基于上一章节构建的电影知识来实现的，因此我们可以查询电影相关的信息，比如电影的剧情简介信息、电影评分、电影上映时间以及电影类型或风格信息。同样我们也可以查询演员的相关信息，比如演员出生日期、演员的国籍信息、演员简介信息等，此外通过电影与演员之间的关系，我们也可以查询某演员出演过那些电影以及某电影有那些演员出演、某演员出演过那些类型风格的电影等信息。

为了提高我们问答系统的用户体验，让其更具智能化，我们可以让我们的问答系统与用户进行简单的问候互动比如理解用户的问候信息及结束问答等。

该问答系统的实现架构图如图6.1所示：



图6.1 基于知识图谱的电影知识问答系统架构图

### 6.1.2 基于知识图谱的电影知识问答系统模块介绍

基于知识图谱的电影知识问答系统主要分为三个功能模块，分别为文本预处理模块、用户意图识别模块与答案查询模块，下面分别介绍下各个功能模块的实现思路。

文本预处理模块：该模块主要完成对用户输入问题的文本预处理，主要包括对输入问题进行文本清理、分词、去除停用词及文本结构化，同时在该模块中对用户输入的问候语及结束问答进行相应识别回复。

用户意图识别模块：该模块主要完成对用户输入问题提取关键信息，如提取电影名或演员名，这里采用jieba分词来实现，根据词性标注信息来抽取关键信息。同时该模块还将使用预训练的分类模型对用户输入的问题进行分类，获取相应的问题模板，最后结合关键信息与问题模板获取用户的意图信息。

答案查询模块：该模块根据用户意图生成相应的知识图谱查询语句，通过对电影知识数据的查询获取最终的答案并返回给用户。

## 6.2 基于知识图谱的电影问答系统

我们仅将数据集中回答为最佳回答的问答数据抽取出来。具体实现代码如下：

**代码4.1 数据预处理代码**

#!/usr/bin/env python

# \_\*\_ coding:utf-8 \_\*\_

import csv

def read\_data():

'''

读取保险数据源文件数据，将其作为数组返回

:return:

'''

# 源数据文件地址

csv\_dir = './baoxianzhidao\_filter.csv'

# 存储读取到的保险问题描述

insurance\_ques = []

# 存储读取到的保险问题对应答案

insurance\_ans = []

# 读取源数据

with open(csv\_dir, 'r', newline='', encoding='utf-8') as csvfile:

csv\_reader = csv.reader(csvfile)

for row in csv\_reader:

# 选取源数据中答案为最优答案的问答数据

if len(row) == 4 and row[3] == '1':

# 若question字段非空，则将问题title字段与question字段拼接为问题描述

# 若question字段为空，则将问题title字段作为问题描述

if row[1]:

insurance\_ques.append(row[0] + row[1])

else:

insurance\_ques.append(row[0])

# 选取replay字段作为答案描述

insurance\_ans.append(row[2])

return insurance\_ques, insurance\_ans

def save\_data(insurance\_ques, insurance\_ans):

'''

存储保险问答数据

:param insurance\_ques:

:param insurance\_ans:

:return:

'''

# 遍历存储问答数据为csv格式

for idx in range(len(insurance\_ans)):

with open('insurance\_data.csv', 'a', newline='', encoding='utf-8') as csvfile:

spamwriter = csv.writer(csvfile)

spamwriter.writerow([insurance\_ques[idx], insurance\_ans[idx]])

if \_\_name\_\_ == '\_\_main\_\_':

insurance\_ques, insurance\_ans = read\_data()

save\_data(insurance\_ques, insurance\_ans)

最终获取到保险问答数据格式如表4.2所示：

表4.2 保险问答数据格式字段介绍

|  |  |  |
| --- | --- | --- |
| 字段 | 说明 | 备注 |
| insurance\_ques, | 保险数据问题 |  |
| insurance\_ans | 保险数据问题相应答案 |  |

insurance\_ques, insurance\_ans

最近在安邦长青树中看到什么豁免，这个是什么意思？,您好，这个是重疾险中给予投保者的一项权利，安\*长青树保障责任规定，投保者可以享受多次赔付，豁免等权益。也就是说不同轻症累计5次赔付，理赔1次轻症豁免后期所交保费，人性化的设计，无需加保费。  
HUTS中有没有适合帆船比赛的保险，我男朋友这周就要开始了,您好，水上运动比赛，尤其是带有奖金的比赛一般承保的公司比较少。不过，HUTS保险中的众行天下-水上运动保险赛事版B就是适合帆船等水上比赛的产品，含户外溺水保障，是水上运动专属定制的保障，意外住院有津贴，保障期限灵活可选，还可以投保有奖金的赛事，您可以根据情况看看。  
计划端午节和男朋友自驾去九\*山，买保险三天要多少钱？,"您好，端午出行的人比较多，而且自驾存在一定风险，所以有保险意识还是很好的。考虑到价格以及保障内容等相关因素，您可以看看HUTS保险中的畅玩神州-慧择旅游保险计划三，适合驾驶私家车走南闯北国内旅游，自驾意外累计赔付,承保的范围也较为广泛，适合带家人出游，保障全面，三天仅需75元，性价比还是蛮高的。"  
端午我们准备要举行赛龙舟，说是要份保险，什么好,您好，赛龙舟是一项比较传统的活动，很有意义。不过由于是在水上活动，建议安全保障还要做足，HUTS保险中有针对水上运动风险特别定制的众行天下-水上运动保险，可以针对这种赛事进行保障，含有意外住院有津贴以及一系列保障，性价比较高，关键是费用也比较实惠。  
老婆买了安\*长\*树，她在网上投保的，以后缴费怎么办,您好，这点是不用担心的。投保后保险公司会在约定的保险费交纳日从消费者购买时填写的银行账号中划扣当期应交的安\*长青树重疾险的保险费，所以您老婆是不用亲自去保险公司缴费的。

## 4.2 基于问题相似度的智能客服实现：智能客服代码实现

为了提高用户体验度，我们将用户的提问分为三种情况处理。

（1）用户提问的问题在问题库中存在，即问题库中存在完全一致或高度相似的问题，那么直接将问题库中该问题的答案返回给用户。

（2）用户提问的问题在问题库中不存在完全一致或高度相似的问题，但存在语义接近的问题，那么将语义最接近问题的答案返回给用户。

（3）用户提问的问题系统无法处理，则通知用户该问题暂时没有找到答案。

其中项目整体架构如图4.4所示：



图4.4 智能客服系统架构

### 4.2.1 生成问题数据向量文件

为了便于我们通过语义向量比较用户输入的问题与问答库中问题的相似度，我们可以预先离线生成问题数据向量文件。

在本部分中，我们将问题描述通过BERT转换为向量表示，并以二进制保存，存储格式为npy文件。

项目中将字符串转换为向量表示的工具为原腾讯AI Lab负责人的HanXiao封装的bert-as-service，项目地址<https://github.com/hanxiao/bert-as-service>，具体使用方法可以参考该项目。

通过转换，将每个问题的描述文本转换为一个长度固定为768的向量，该向量蕴含表示了该问题的语义向量，方便我们下一步使用。

具体代码如下：

**代码4.2 将问题数据转换为向量表示**

#!/usr/bin/env python

# \_\*\_ coding:utf-8 \_\*\_

import numpy as np

from bert\_serving.client import BertClient

import csv

def get\_insurance\_question():

'''

获取问答数据中的问题

:return:

'''

# 保险问答数据存储地址

source\_dir = './insurance\_data.csv'

# 存储保险问答数据中的全部问题

insurance\_question = []

# 读取保险问答数据中的问题字段信息

with open(source\_dir, 'r', encoding='utf-8') as csvfile:

read = csv.reader(csvfile)

for data in read:

insurance\_question.append(data[0])

print('获取到', len(insurance\_question), '条问题')

return insurance\_question

def bertconvert(insurance\_question):

'''

通过BERT将问题转换为向量表示

:param insurance\_question:

:return:

'''

question\_list = []

# 调用bert-as-service将字符串转换为向量表示

bc = BertClient()

for i in range(0, len(insurance\_question)):

curr\_ques = insurance\_question[i]

# 清楚问题字符串中的空格

curr\_ques = "".join(curr\_ques.split())

question\_list.append(curr\_ques)

# 将问题字符串转换为向量表示

insurance\_ques\_vector = bc.encode(question\_list)

# 将向量表示的数据保存为npy格式

np.save("insurance\_ques\_vector.npy", insurance\_ques\_vector)

if \_\_name\_\_ == '\_\_main\_\_':

print('将保险数据中问答数据中的问题生成向量文件...')

insurance\_question = get\_insurance\_question()

bertconvert(insurance\_question)

print('生成向量文件结束！')

### 4.2.2 在问答数据库中查找相同问题

为了判定用户输入的问题是否在问答数据库中存在相同问题及高度相似的问题，我们采用编辑距离来评估用户输入的问题及问答库中问题的相似度，通过Python的Levenshtein库来计算两个字符串的编辑距离相似度。

**代码4.3 将查找相同问题**

def getSameQuestionByEditDistance(curr\_ques):

'''

根据编辑距离计算问题库中相同问题，返回相似值及索引

:param curr\_ques:当前用户输入的问题

:return:

'''

same\_question\_threshold = 0.98

# 保险问答数据地址

insurance\_data\_path = './../preprocess\_data/insurance\_data.csv'

# 最大相似值

max\_similarity\_val = 0

# 最相似问题的索引值

max\_similarity\_index = 0

# 读取保险问答数据

with open(insurance\_data\_path, 'r', encoding='utf-8') as csvfile:

ques\_ans = csv.reader(csvfile)

# 当前比较的数据在问答数据中的索引值

curr\_idx = 0

# 遍历计算用户输入问题与问答题库中全部问题的编辑距离相似度

for curr in ques\_ans:

curr\_idx += 1

# 去除待比较计算问题文本中的标点符号

curr\_ques = replace\_punctuation(curr\_ques)

csv\_ques = replace\_punctuation(curr[0])

# 计算待比较的问题文本的编辑距离的相似度

edit\_distance\_val = Levenshtein.ratio(csv\_ques, curr\_ques)

# 若当前获取的相似度大于已知的最大相似度，则更新最大相似度为当前相似度的值，更新最相似问题的索引为当前索引

if edit\_distance\_val > max\_similarity\_val:

max\_similarity\_val = edit\_distance\_val

max\_similarity\_index = curr\_idx

# 是否存在相同问题的标志位

is\_exist\_same\_ques = False

# 系统回复信息

sys\_reply = ''

# 找到的问题信息

QA\_que = ''

# 找到的答案信息

QA\_ans = ''

# 若查找到的最相似问题的最大相似度大于设置的阈值，则返回该问题及对应答案

if max\_similarity\_val > same\_question\_threshold:

similaryQuestion, bestAns = getSimilaryQuestionByIndex(max\_similarity\_index)

# 答案信息清理

QA\_ans = clean\_ans(bestAns)

is\_exist\_same\_ques = True

return is\_exist\_same\_ques, sys\_reply, QA\_que, QA\_ans

**代码4.4 去除文本中的标点符号**

def replace\_punctuation(curr\_string):

'''

清理无效标点符号

:param curr\_string:当前待处理字符串

:return:

'''

# 通过正则表达式清理中文标点符号

punctuation = "！？｡＂＃＄％＆＇（）＊＋，－／：；＜＝＞＠［＼］＾＿｀｛｜｝～｟｠｢｣､、〃》「」『』【】〔〕〖〗〘〙〚〛〜〝〞〟〰〾〿–—‘'‛“”„‟…‧﹏."

re\_punctuation = "[{}]+".format(punctuation)

curr\_string = re.sub(re\_punctuation, "", curr\_string)

# 通过正则表达式清理英文标点符号

punctuation2 = '!"#$%&\'()\*+,-./:;<=>?@[\\]^\_`{|}~'

re\_punctuation2 = "[{}]+".format(punctuation2)

curr\_string = re.sub(re\_punctuation2, "", curr\_string)

return curr\_string

### 4.2.3 在问答数据库中查找相似问题

当用户输入的问题在问答数据库中查找不到相同问题，则查找与用户输入问题语义相似的问题，首先将用户输入的问题通过BERT转换为向量表示，然后与问答数据的问题向量进行比较，选取相似问题。向量之间相似度的比较我们采用余弦相似度计算方法。

注意：这里需要设置一个阈值，当用户输入问题与问答题库中全部问题的语义相似度均小于该阈值，则表示用户输入问题在问答库中没有相应答案。

具体代码如下：

**代码4.5 查找相似问题**

def getBestAnswer(input\_ques):

'''

根据用户输入问题，系统给予用户相应回答

:param input\_ques:用户输入问题信息

:return:

'''

# 根据编辑距离计算问题库中是否存在相同问题

is\_exist\_same\_ques, sys\_reply, QA\_que, QA\_ans = getSameQuestionByEditDistance(input\_ques)

if is\_exist\_same\_ques:

return sys\_reply, QA\_que, QA\_ans

else:

# 保险问答数据中问题的向量表示数据地址

insurance\_data\_vector\_path = './../preprocess\_data/insurance\_ques\_vector.npy'

# 导入问答数据中问题的向量表述数据

insurance\_ques\_vector = np.load(insurance\_data\_vector\_path)

# 实例化BERT向量转换工具

bc = BertClient()

# 将用户输入问题通过BERT转换为向量表示

input\_vec = bc.encode(["".join(input\_ques.split())])

# 查找问答库中语义最相似的前10个问题

topk = 10

# 计算输入问题与问答数据中全部问题的相似度

score = np.sum(input\_vec \* insurance\_ques\_vector, axis=1) / np.linalg.norm(insurance\_ques\_vector, axis=1)

# 对相似计算结果排序并返回前10条问题

topk\_idx = np.argsort(score)[::-1][:topk]

# 根据找到的最相似问题的索引获取其相似问题及答案

similaryQuestion, bestAns = getSimilaryQuestionByIndex(topk\_idx[0] + 1)

# 计算找到的语义最相似问题与用户输入问题的余弦相似度

similar\_val = cosine\_similarity(input\_vec[0], insurance\_ques\_vector[topk\_idx[0]])

# 用户输入问题与问答库中找到的相似问题的阈值设定

similarity\_question\_consia\_threshold = 0.9

# 若用户输入问题与找到的最相似问题的相似度小于设定的阈值，则表示系统没有找到答案

# 否则将找到的相似问题及答案返回给用户

if similar\_val < similarity\_question\_consia\_threshold:

sys\_reply = '啊哦，小助手还没有掌握这方面的知识呢，我会将您的问题记录下来，并尽快找到专业的答案。'

QA\_que, QA\_ans = '', ''

return sys\_reply, QA\_que, QA\_ans

else:

sys\_reply = '小助手没有这个问题的答案呢，给您推荐以下相似问题及答案以供参考哦~\n'

QA\_que = '相似问题：' + similaryQuestion + '\n'

QA\_ans = '推荐答案：' + bestAns + '\n'

QA\_ans = clean\_ans(QA\_ans)

return sys\_reply, QA\_que, QA\_ans

**代码4.6 计算向量余弦相似度**

def cosine\_similarity(vector1, vector2):

'''

计算余弦相似度

:param vector1:待比较向量1

:param vector2:待比较向量2

:return:

'''

dot\_product = 0.0

normA = 0.0

normB = 0.0

# 根据余弦相似度计算公式计算两个向量的余弦相似度

for a, b in zip(vector1, vector2):

dot\_product += a \* b

normA += a \*\* 2

normB += b \*\* 2

if normA == 0.0 or normB == 0.0:

return 0

else:

return round(dot\_product / ((normA \*\* 0.5) \* (normB \*\* 0.5)), 2)

**代码4.7 根据问题索引查找问题及相应答案信息**

#!/usr/bin/env python

def getSimilaryQuestionByIndex(index):

'''

根据问答数据的问题索引获取其答案信息，返回问答数据中找到问题及答案信息

:param index:

:return:

'''

# 问答数据地址

insurance\_data\_path = './../preprocess\_data/insurance\_data.csv'

# 读取问答数据

with open(insurance\_data\_path, 'r', encoding='utf-8') as csvfile:

read = csv.reader(csvfile)

idx = 0

# 遍历问答数据

for curr\_data in read:

idx += 1

# 根据问题索引查找其答案信息

if idx == index:

curr\_ques = curr\_data[0]

curr\_ans = curr\_data[1]

return curr\_ques, curr\_ans

**代码4.8 答案信息清理**

def clean\_ans(str\_ans):

'''

清理答案信息中的不合法字符

:param str\_ans:

:return:

'''

# 清理空格及换行字符

str\_ans = str\_ans.replace('\n', '').replace(' ', '')

return str\_ans

项目实际测试效果如下：

当前用户输入问题为： 天热了，很想去潜水，除了装备、教练以及费用外还要什么保险？

当前智能客服回复为： 您好！除了您所说的准备、教练等专业人士陪同之外，购买一份合适的户外运动保险很有必要，毕竟潜水存在一定风险。目前国内关于潜水运动的保险并不多，一般都要求是下潜不超过18m。具体您可以了解下HUTS保险中的适合国外潜水等海岛相关水上运动的产品众行天下-水上运动保险，是海岛潜水专属的保障。主要是因为它同时包括潜水意外医疗补偿以及紧急医疗运送，是高风险水上运动的理想选择。

------------------------------------------------------

当前用户输入问题为： 本周末公司组织一次大型户外拓展活动，100人什么HUTS保险合适？

当前智能客服回复为： 小助手没有这个问题的答案呢，给您推荐相似问题及答案以供参考哦~ 本周末公司会组织一次大型户外拓展活动，200人什么HUTS保险合适？ 您好，200人的户外拓展规模是比较大的，相对而言，也存在一定的风险，所以建议还是要考虑众行天下-拓展训练保险。这是款适合集体投保的，最高可赔付10万元，关键是价格比较实惠。

------------------------------------------------------

当前用户输入问题为： 太阳系有几个行星呢

当前智能客服回复为： 啊哦，小助手还没有掌握这方面的知识呢，我会将您的问题记录下来，并尽快找到专业的答案。

### 4.2.4 封装API并部署在服务器

为了便于访问我们的智能客服服务，我们将我们的代码封装成一个API，并部署在服务器上，是的通过http请求即可访问获取。这里采用aiohttp来实现服务部署。相关知识可以访问：<https://docs.aiohttp.org/en/stable/>

智能客服服务部署在我们本地，接口设置为9010，故访问地址为：<http://127.0.0.1:9010>

通过GET请求进行访问，参数为question

**代码4.9 封装API及项目部署代码**

#!/usr/bin/env python  
# \_\*\_ coding:utf-8 \_\*\_  
from aiohttp import web  
import asyncio  
from intelligent\_service import getBestAnswer  
import time  
  
  
async def handle(request):  
 '''  
 处理GET请求  
 :param request:  
 :return:  
 '''  
 varDict = request.query  
 question = varDict['question']  
 question = clean\_text(question)  
 sys\_reply, QA\_que, QA\_ans = getBestAnswer(question) # 比较余弦相似度查找相似问题  
 reply\_json = {'系统回复': sys\_reply, '相似问题': QA\_que, '推荐答案': QA\_ans}  
 print('Current time is:', time.strftime('%Y-%m-%d %H:%M:%S', time.localtime(time.time())))  
 return web.json\_response(reply\_json)  
  
  
def clean\_text(word):  
 '''  
 问题文本清理  
 :param word:  
 :return:  
 '''  
 res = ''  
 for ch in word:  
 if '\u4e00' <= ch <= '\u9fff':  
 res += ch  
 return res  
  
  
async def init\_app():  
 app = web.Application()  
 app.router.add\_get('/', handle)  
 return app  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 loop = asyncio.get\_event\_loop()  
 app = loop.run\_until\_complete(init\_app())  
 web.run\_app(app, port='9010')

项目整体代码在如下地址提供：xxx

### 4.2.5 智能客服效果展示

服务部署以后，我们通过在浏览器输入想查询的问题，系统会给出相应回复。问题数据通过在访问地址中拼接?question=XXX的方式传递。

如：

案例1：

<http://127.0.0.1:9010/?question=>天热了，很想去潜水，除了装备、教练以及费用外还要什么保险？

{

"系统回复": "",

"相似问题": "",

"推荐答案": "您好！除了您所说的准备、教练等专业人士陪同之外，购买一份合适的户外运动保险很有必要，毕竟潜水存在一定风险。目前国内关于潜水运动的保险并不多，一般都要求是下潜不超过18m。具体您可以了解下HUTS保险中的适合国外潜水等海岛相关水上运动的产品众行天下-水上运动保险，是海岛潜水专属的保障。主要是因为它同时包括潜水意外医疗补偿以及紧急医疗运送，是高风险水上运动的理想选择。"

}

案例2：

<http://127.0.0.1:9010/?question=>本周末公司组织一次大型户外拓展活动，100人什么HUTS保险合适？

{

"系统回复": "小助手没有这个问题的答案呢，给您推荐以下相似问题及答案以供参考哦~\n",

"相似问题": "相似问题：本周末公司会组织200人的户外拓展活动，什么保险能保障员工安全？\n",

"推荐答案": "推荐答案：您好，关于户外拓展活动，存在一定的风险，并且贵公司有200人参加，所以建议可以考虑下HUTS保险中的众行天下-拓展训练保险，是专门针对户外运动这块的保险。最高可赔付10万元，关键是价格比较实惠，适合集体投保。"

}

案例3：

<http://127.0.0.1:9010/?question=>太阳系有几个行星呢

{

"系统回复": "啊哦，小助手还没有掌握这方面的知识呢，我会将您的问题记录下来，并尽快找到专业的答案。",

"相似问题": "",

"推荐答案": ""

}

## 4.3 小结

通过上面的实践，我们已经搭建了一个简单的智能客服系统。为了让这个系统更完善，我们可以通过以下几个方面进一步完善。

（1）我们可以再设计一个漂亮的前端页面，便于与用户交互。

（2）将我们的服务应用在移动端，或者小程序等，扩大客服的应用场景。

（3）在功能上，为了提升系统对问题的处理能力，可以不断扩充我们的问答数据库。

（4）不断优化算法，提升文本匹配精度，帮助用户更准确到查找到语义相近的问题。

（5）我们也可以尝试使用分词的其他词向量技术如word2vec来比较问题文本的的相似度。

（6）这里对文本相似度的比较采用了计算余弦相似度度量方式，我们还可以采用上一章节介绍的基于pairwise及pointwise的方式来比较文本的相似度。