复旦美食问答系统

18307130264 陶静怡

摘要：去食堂吃什么是复旦同学每天都会面临的问题，对于食堂有什么美食也是我们很关注的问题。对于校外的就餐我们可以很方便地在各大评论网站上找到一些评价信息，但是我们在学校就餐的时候很难得到这样的评价信息。本项目通过收集大众点评上的一些对复旦食堂的评价信息和周围同学对菜品的推荐和吐槽，基于自然语言处理中的文本分类、情感分析、分词和关键词抽取等技术，建立了复旦美食问答系统。用户可以提问关于校区、食堂的各类菜品信息和具体菜品的评价情况。

1. 选题动机

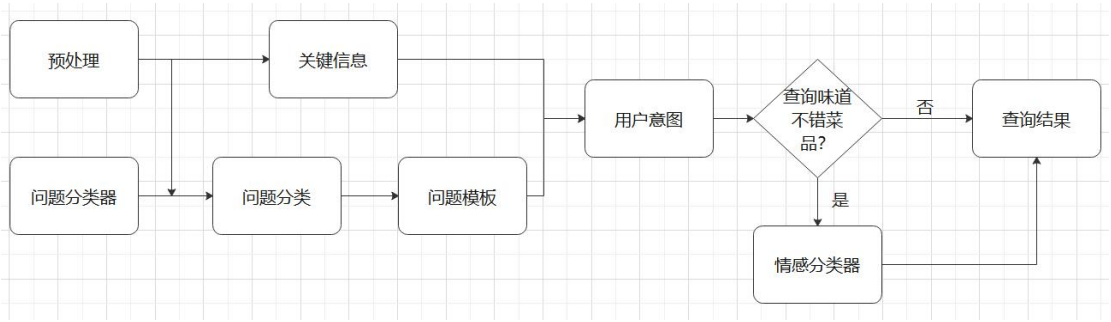
现在我们在外面就餐的时候都喜欢在各大点评网站上寻找优质的餐厅，通过别人的评价来挑选餐厅，选择菜品。但是我们在学校就餐的时候很难得到这样的评价信息，一般信息来源只有通过同学的推荐和吐槽，而且同学之间也不太讨论食堂里的菜，再加上我们的食堂不像外面的饭店会通过微信公众号等形式来做广告宣传或者是办促销活动来吸引顾客，食堂里有时候出了新的菜品我们也不太了解，而且在就餐人数比较多的时候，如果是吃大众菜的话一般快排到自己了才能看看有什么菜，有时候随便点很容易踩雷。此外，虽然在大众点评上可以得到一些对于复旦食堂的评价，但是评价内容比较模糊，包含很多不相关的信息，而且一条评价里往往有对多个菜品的点评，同时褒贬不一，查询起来很麻烦，而且通常同学也不会在去食堂吃饭的时候还特意去大众点评上找信息。

因此，针对很多同学在食堂就餐的时候经常有今天吃什么的困扰这一现象，希望建立一个复旦美食问答系统，让同学在去食堂的路上或是排队的时候就可以通过这个系统轻松获取信息，不用再为吃什么而伤脑筋。

本项目旨在解决查询复旦食堂美食时的不便，通过对话的方式，获得用户的查询问题后进行分析，将结果用文字问答的方式提供给用户。同时设想之后同学可以把评价提交到系统中，食堂也可以通过这个系统来进行菜品的宣传。

2.结构

本系统的结构如图所示:



(1)预处理

对用户的问题进行预处理的过程中，需要对句子进行命名实体识别，知道问题中的主语是菜品、食堂名字、校区名字还是菜系。然后需要知道用户询问的是哪类问题，和问题分类器结合得出问题的类型。

(2)问题分类器

训练一个问题分类器来知道用户询问的是哪类问题，用户输入问题可以对应到问题的类别。这里用了朴素贝叶斯的模型。

(3)问题模板

在训练问题分类器中把问题分成多类，比如某个食堂有什么菜，某个食堂的人均，菜品的评价等等。对各个问题抽象成模板的形式，有助于对问题意图的理解和对关键信息的提取。比如询问某个食堂的人均可以把它抽象成"rr 人均"，其中 rr 是对食堂名的词性标注。

(5)关键信息

对于关键信息抽取中的命名实体识别，采用词性标注的方法提取。增加对食堂名，校区名等的标注，可以把关键信息抽取出来。同时增加词性标注以后还可以增加问题分类器的准确性。

(4)查询答案

在知道用户的意图之后，就可以进行查询，得到结果后返回，这里主要是对数据库进行查询，这里用的 mysql 数据库。这里还有区分问题是否和情感分类有关，如果是与情感分类相关的问题如 xx 食堂味道不错的菜，还需要对查询结果中的菜品评价使用一个预训练的情感分类器，得出菜品的好坏，返回情感倾向积极的菜品。

3.数据获取

本项目中使用了 mysql 数据库，储存信息包括食堂，校区，菜品，菜系，评价，人均等。

3.1 基础数据导入

本项目中的菜品评价信息主要来自大众点评，一部分来自周围同学的评价。选择了邯郸校区的南区，北区，本部中的 7 个食堂，共 54 种菜品，15 个菜系，并对每一种菜做出文字评价。

对于大众点评上的评价使用 Beautiful Soup 和 Re 模块进行页面的解析，得到评价的数据。由于对于我们学校食堂的评价比较少，而且很多是过期的信息，很多评价里有很多和菜品无关的内容。

因为网上对于复旦食堂的评论较少，而且评论内容都比较乱，所以数据集较小，但由于重点在于自然语言处理的部分，数据集只和最后一部分查询内容相关。而且后续可以考虑让同学自己加入评论使数据集扩大。

文件名	储存内容	属性	数量
dish.csv	菜品	菜品 id,食堂 id,菜名评价，菜系	54
restaurant.csv	食堂	食堂 id,名称,评分，人均校区名称	7

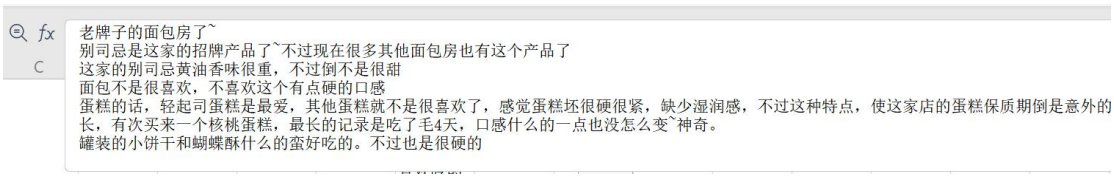
导入 mysql 数据库后如图所示：

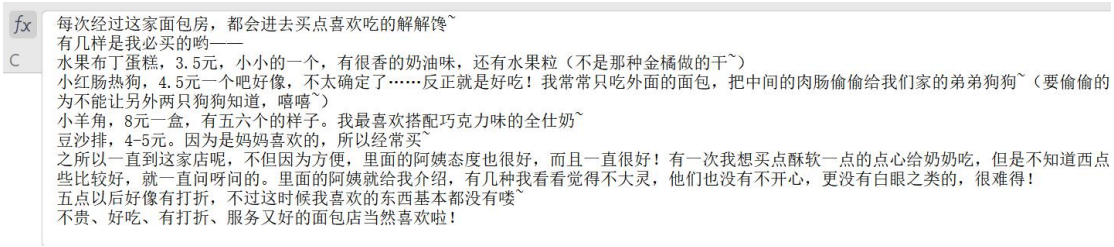
cid	rid	name	review	type
37	4	铁板菜	现点现做，味道不f	粤菜
38	4	水饺	可以选馅，但是味i	面点
39	5	烧烤	虽然味道一般，但i	烧烤
40	5	重庆小面	现在换师傅了，水i	重庆菜
41	5	酒酿圆子	酒酿味道不浓，偏i	本帮菜
42	5	石锅拌饭	一楼的石锅拌饭真i	韩式料理
43	5	大盘鸡面	分量很足，里面的i	新疆菜
44	5	小火锅	里面有各种丸子，i	火锅
45	7	小笼	太优秀了，相见恨i	点心
46	7	红豆薏米粥	心中的北食no.1,4.	早点
47	7	肠粉	肠粉皮整体表现很i	粤菜
48	7	蒸饺	猪肉大葱蒸饺，9i	面点
49	7	糯米肉丸	很好吃，但是如果i	本帮菜

rid	name	score	cost	campus
1	旦苑	4.2	10	本部
2	复旦点心部	4	11	本部
3	南食	3.9	12	南区
4	南小食	4.3	14	经院
5	春晖	3.8	18	南区
6	南苑	4.3	21	南区
7	北食	4.3	22	北区

3.2 情感分类器训练集

由于对于复旦的菜品评价信息比较少，所以选择爬取点评网站上对美食的评价信息。由于点评网站反爬比较厉害，而且得到的很多评价信息内容和评分不符，或者是内容模糊。





此外很多评价都是综合了排队时长，食堂卫生等等情况对店家的评价，但是我们需要的只是对单一菜品的评价信息。所以在得到的一千多条评价中去除一些无用的信息，同时其中还包含一些噪音数据，训练的模型容易发生过拟合，还有的评价情感取向比较模糊，难以判断其归类。所以最后通过人工标注数据集中的文本情感（1 代表积极，0 代表消极），最终删选出两百多条有效的评价文本，部分评价如下图:

25	这个是惊喜，酸酸甜甜的，清爽可口，一定要试	1
26	原味的就可以了，现烤出来的皮酥酥的，内馅味	1
27	每次路过这边，都会被那股浓浓又甜甜的奶香给	1
28	口感真的是超级棒，感觉入口即化了，真的是香	1
29	三人吃了四个菜，盘盘清光，味道很不错	1
30	以前去都挺好的。但是最近去发现做的越来越差	0
31	好像没有什么特色，没有特别好吃，也不正宗	0
32	比较正点的！！东西辣得够爽	1
33	冬天十来个朋友聚会，那里是个不错的选择，消	1
34	分量足，经常是吃剩3分之一的。而且炒出来的菜	1
35	不怎么辣。分量倒是挺大的。总得来说担得起物	1
36	菜难吃不是厨师错，可是他每个菜都放那么多盐	0
37	这里才也实在太贵了吧，主要是性价比十分十分	0
38	味道实在实在不想说了，还是吃路边摊吧	0
39	满满得一锅，味道确实不错，偏辣偏咸，对我得	1
40	番茄酱的，不是我喜欢的口	0
41	比较适合大家聚餐来吃，真叫一个热火朝天啊	1
42	怎么说呢，吃起来真的是没有想象的那么好吃，	0

4.构建系统

4.1 问题与模板设计

在系统中自定义词性标注如下:

```
# 词性标注的构建
# nm   菜名
# ut 校区名
# rr 食堂名
# sst 菜系名
```

提供的问题如下:

（1）菜品评价
（2）某食堂的主要菜品
（3）某校区的食堂
（4）某食堂的人均价格
（5）某食堂的评分
（6）某校区味道不错的菜
（7）某食堂味道不错的菜
（8）某菜系有什么菜

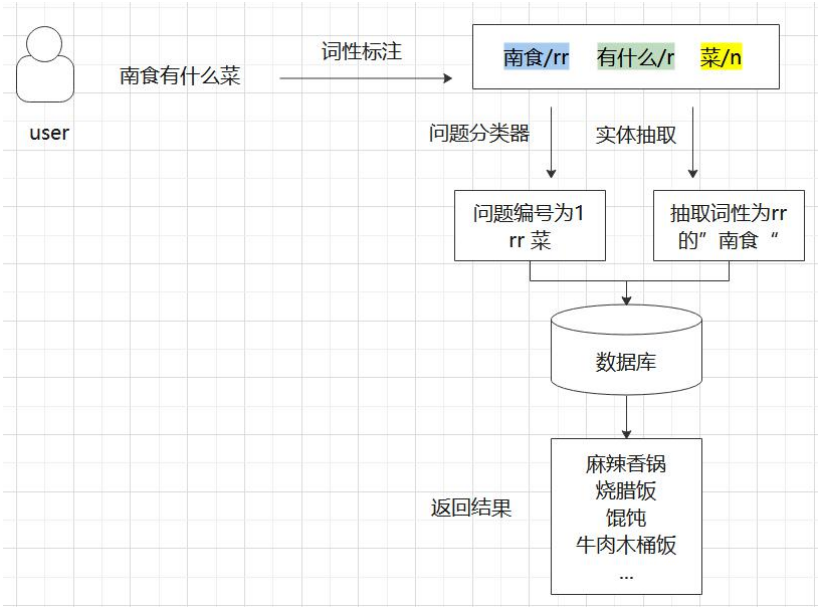
共八类问题，对应的模板如下:

```
questions = {
    0: "nm 味道",  # 菜品评价
    1: "rr 菜",  # 某食堂的主要菜品
    2: "ut 食堂",  # 某校区的食堂
    3: "rr 人均",  # 某食堂的人均
    4: "rr 评价",  # 某食堂的评分
    5: "rr 好吃的菜",  # 某个食堂的味道不错的菜
    6: "ut 好吃的菜",  # 某个校区的味道不错的菜
    7: "sst 菜",  # 某菜系有什么菜
}
```

后续还可以增加更多的问题来拓展系统的功能。

4.2 推理引擎和具体算法

对于用户提出的问题，工作的步骤是:首先对于原始问题进行词性标注，实体提取，之后将原始问题转化为部分实体以标注的词性替代的问题。对于该问题，通过训练一个问题分类器来识别该问题对应的问题模板，最终在数据库中执行查询操作，将结果返回。



4.2.1 基于朴素贝叶斯方法的问题分类器训练

提供了八类问题，对每一类问题，有不同的问法，考虑建立一个问题分类器来识别用户的问题。该分类器的训练数据是每一类问题的各种问法，标签是对应的问题号。例如问题 0 对饮的是某个菜的评价，提供的训练数据如下:

```
# Question - 0 : 菜品评价
q0 = [
    "nm好吃吗",
    "nm味道怎么样"
    "nm有人推荐吗",
    "nm好不好吃",
    "nm怎么样",
    "nm的评价",
    "nm评价",
]
```

这样对于八类问题，每个问题的输入是这个问题的所有可能的问法。得到了一组带标注的训练数据，用朴素贝叶斯分类器来问题分类。

使用 TF-IDF，把句子变成向量，以组成矩阵来输入到模型中

```
def train_with_mulNB(self):
    print(self.x)
    print(self.y)
    x_train, y_train = self.x, self.y
    self.tv = TfidfVectorizer()
    train_data = self.tv.fit_transform(x_train).toarray()
```

使用多项式朴素贝叶斯 MultinomialNB

```
mms = MultinomialNB(alpha=0.01)
mms.fit(train_data, y_train)
```

这里把平滑的参数λ设置为 0.01。平滑相当于人为给概率加上一些噪音，因此λ设置得越大，多项式朴素贝叶斯的精确性会越低，所以这里设置的比较小，而不是用默认的 1.

4.2.2 基于 SnowNLP 进行情感分类

由于提供的问题中还有查询味道不错的菜的功能，因此要判断哪些菜是味道不错的，因此需要一个情感分类器。这里考虑预训练一个情感分类器，用标注的菜品评价信息作为训练集，训练出一个对菜品评价的情感分类器。然后就可以用这个分类器来判断菜品是否属于“味道不错”的范畴，将这个菜的评价信息输入到预训练好的情感分类器中，输入的结果越接近 1 则评价越高。

在这里中，使用到情感分类器模型是在筛选出味道不错的菜，所以每次用户查询的时候会对数据库中的菜品评价信息进行一次情感分类，返回情感倾向积极的菜品，所以使用到的预训练模型计算速度要快。深度学习模型如 RNN 等是时序模型，不能进行并行计算，速度比较慢，所以这里选择用传统的机器学习方法进行情感分类，虽然准确度稍低，但是模型的计算速度很快。当然后续可以考虑把菜品评价的情感倾向在用户把评价存入数据库的时候就分析好，把这个信息也存在数据库中，就不用每次都去计算了。

这里使用了 SnowNLP 来进行情感分类，首先把训练集存入 neg.txt 和 pos.txt 两个文件中：

```
neg , pos = "","" 
csv_file = open("review.csv")
csv_reader_lines = csv.reader(csv_file)
i = 0
pos_count ,neg_count = 0,0

for line in csv_reader_lines:
    content, label = line[0] , line[1]
    if line[1] == '0':
        words = jieba.cut(content)
        neg += ' '.join(jieba.cut(content))
        neg_count += 1
    if line[1] == '1':
        pos += ' '.join(jieba.cut(content))
        pos_count += 1

with open("pos.txt", "w", encoding='utf-8') as f:
    f.write(pos)

with open("neg.txt", "w", encoding='utf-8') as f:
    f.write(neg)
```

然后使用了 SnowNLP 进行情感分类，代码如下：

```
sentiment.train('neg.txt','pos.txt')
sentiment.save('sentiment.marshal')
```

同时分别对正负样本做了词云分析，结果如下：左边是负样本，右边是正样本：
可以看到评价比较高的菜品，主要是味道不错，喜欢，可以等等，而评价一般的菜品，主要特点是一般，贵，太咸等等。

4.2.3 分词与问题匹配

对于菜名，校区名，菜系名和食堂名都自定义了词性类型，所以在用 jieba 进行词性标注之前要加入这些标注词的信息，部分标注如下图所示。

南食 15 rr
南小食 15 rr
春晖 15 rr
南苑 15 rr
北食 15 rr
点心 15 sst
面点 15 sst
韩式料理 15 sst
川菜 15 sst
日式料理 15 sst
本帮菜 15 sst
陕菜 15 sst
早点 15 sst
汤 15 sst
湖南菜 15 sst
重庆菜 15 sst

然后对于原始的问题，采用了 jieba 进行分词，得到这句话的词性标注。把其中的部分信息转换为词性标注（eg.把南食替换为标注 rr）,将得到的新的问题输入问题分类器，输出为该问题对应的问题模板 0。

4.2.4 数据库查询

对每一类问题的查询方法如下:

问题号	内容	查询方法
0	菜品的评价	查询菜名为用户输入的菜品，返回其评价内容
1	某食堂的主要菜品	查询该食堂的菜品，返回所有菜名
2	某校区的食堂	查询该校区内的食堂，返回所有食堂名
3	某食堂的人均	查询该食堂的人均消费，返回价格
4	某食堂的评分	查询该食堂的评分，返回评分
5	某个校区味道不错的菜	首先查询这个校区下的食堂，再查询这个食堂下的菜品，使用预训练模型计算这个菜的评价内容的情感得分，取大于 0.5 的为结果
6	某个食堂味道不错的菜	查询这个食堂下的菜品，使用预训练模型计算这个菜的评价内容的情感得分，取大于 0.5 的为结果
7	某菜系的菜	查询这个菜系下的菜，返回对应的菜品和所在的食堂

在查询的结果后加上一些修饰语，组成最后的返回结果。

下面列举一些查询的语句:

（查询菜品的评价）

```
sql = "select review from dish where name=%s"
answer = self.query.run(sql, dish_title)[0][0]
```

（查询某食堂的主要菜品）

```
sql = "select dish.name from dish,restaurant where restaurant.name=%s and dish.rid = restaurant.rid"
ans_list = self.query.run(sql, restaurant)
```

5 实验过程和结果

5.1 问题分类器实验结果

在构建完问题分类器中的训练数据集以后，测试该分类器的预测结果。

```
5  from classifyq import *
```

```
9  que = Question_classify()
   true_ans = [0,0,1,1,4,6,2,3,2,5]
   questions = [
       "双皮奶好吃吗",
       "红烧肉味道怎么样",
       "南食有什么吃的",
       "去北食吃什么",
       "春晖怎么样",
       "旦苑推荐的菜",
       "南区有什么食堂",
       "北食人均多少",
       "北区有什么食堂",
       "南区推荐的菜",
   ]
```

```
['nm 好吃 吗', 'nm 味道 怎么样 nm 有人 推荐 吗', 'nm 好不好 吃', 'nm 怎么样', 'nm 的 评价', 'nm 评价',
[0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3, 4, 4, 5, 5, 5, 5, 5, 5, 6, 6, 6, 6,
```

由于问题的类别数目比较少，而且每一类问题之间的差异比较明显，所以用朴素贝叶斯分类器可以训练出不错的结果，对用户的提问能准确判断。

```
10 count = 0
   for question in questions:
       res = que.predict(question)
       idx = questions.index(question)
       if res == true_ans[idx]:
           count+= 1
           print("问题",idx+1,"正确判断")
       else :
           print("问题",idx+1,"错误判断")

   print("正确判断的比例为:",str(count/len(true_ans)))
```

```
问题 1 错误判断
问题 2 正确判断
问题 3 正确判断
问题 4 正确判断
问题 5 正确判断
问题 6 正确判断
问题 7 正确判断
问题 8 正确判断
问题 9 正确判断
问题 10 错误判断
正确判断的比例为: 0.8
```

可以看到用朴素贝叶斯分类器的模型拟合结果还可以，基本可以学习到每个问题中的语言模型，但是没有达到要求。对于问题 1 和问题 10，即某个菜品的评价和某个校区的推荐菜，该模型不太能准确分辨，因此后续用了词性标注来替换原问题来进行预测，在 3.3.3 中有提及，下图是用词性标注替换后的预测结果，可以看到预测正确。

```
print(que.predict("nm好吃吗"))
print(que.predict("ut推荐的菜"))
```

```
0
5
```

5.2 情感分类器实验结果

由于 SnowNLP 包中内置了情感分析工具，对同一句话（不在训练集中的）分别运用原包中内置的情感分析工具和通过评价文本评价训练后的情感分析模型，可以看到运用内置的模型对这句化的情感得分很低，偏向消极求序，比较准确。

```
24 sentiment.train('dish_review/neg.txt','dish_review/pos.txt')
   print(sentiment.classify('太咸了，而且很贵'))

   0.06953978894746837
```

下图是训练好的模型对菜品评价的运行结果，这几个例子都是不在训练集中。可以看到，对于正面评价，情感得分接近 1，而对于负面评价，得分接近于 0。这说明我们训练的情感分析模型较为有效。

```
pos_reviews = [
    "现点现做，味道不错",
    "味道很好，色香味俱全，看上去就很有食欲",
    "分量足，经常是吃完剩很多。而且菜都很好看，让人看了就大动食指"
]
for review in pos_reviews:
    print(SnowNLP(review).sentiments)

0.9254977070840255
0.9970918986332429
0.9093161495206773
```

```
36 neg_reviews = [
       "没有什么特色，也不正宗",
       "真的有点后悔来这里吃，太差劲了",
       "本来很是期待，结果吃了之后，大失所望啊",
   ]
   for review in neg_reviews:
       print(SnowNLP(review).sentiments)

   0.07457891596677191
   0.06245225458137027
   0.13903283091487206
```

5.3 查询运行结果

前面的部分对系统的各个部件都进行了说明，把各部分进行组装之后，即把词性标注，问题分类，实体抽取，问题匹配和查询结果结合起来就可以工作。

下面会展示各类查询的运行结果：

问题 0-nm 味道 （对菜品的评价）：

```
que = Question()
result = que.process_question("双皮奶味道怎么样")
print(que.question_posseg())
print("查询结果：",result)
```

```
Building prefix dict from the default dictionary ...
Loading model from cache C:\Users\tjy\AppData\Local\Temp\jieba.cache
Loading model cost 0.755 seconds.
Prefix dict has been built successfully.
['nm 好吃 吗', 'nm 味道 怎么样 nm 有人 推荐 吗', 'nm 好不好 吃', 'nm 怎么样', 'nm 的 评
[0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3, 4, 4, 5, 5, 5, 5,
抽象问题为: nm味道怎么样
使用模版编号： 0
问题模版： nm 味道
['双皮奶/nm', '味道/n', '怎么样/r']
查询结果： 双皮奶评价:有网红双皮奶卖，复旦学子有口福了~
```

问题 1- rr 菜 （某食堂的主要菜品）

问题1- rr 菜 （某食堂的主要菜品）

```
result = que.process_question("南食有什么吃的")
print(que.question_posseg())
print("查询结果：",result)
```

```
抽象问题为: rr有什么吃的
使用模版编号： 1
问题模版： rr 菜
['南食/rr', '有/v', '什么/r', '吃/v', '的/uj']
查询结果： 南食的菜有：麻辣香锅、烧腊饭、馄饨、牛肉木桶饭、糖醋排条、红烧肉、芝麻包、菠萝古老肉、瓦罐汤...
```

问题 2- ut 食堂, （某校区有哪些食堂）

问题2- ut 食堂,（某校区有哪些食堂）

```
result = que.process_question("南区有什么食堂")
print(que.question_posseg())
print("查询结果: ",result)
```

```
抽象问题为: ut有什么食堂
使用模版编号： 2
问题模版： ut 食堂
南区
['南区/ut', '有/v', '什么/r', '食堂/n']
查询结果： 南区的食堂有：南食、春晖、南苑~
```

问题 3- rr 人均, （某食堂的人均）

问题3- rr 人均,（某食堂的人均）

```
result = que.process_question("北食人均多少")
print(que.question_posseg())
print("查询结果: ",result)
```

```
抽象问题为: rr人均多少
使用模版编号： 3
问题模版： rr 人均
['北食/rr', '人均/j', '多少/m']
查询结果： 北食人均22元~
```

问题 4-rr 评价，（某食堂的评分）

问题 4-rr 评价，（某食堂的评分）

```
result = que.process_question("旦苑评价怎么样")
print(que.question_posseg())
print("查询结果: ",result)
```

```
抽象问题为: rr评价怎么样
使用模版编号：  4
问题模版： rr 评价
['旦苑/rr', '评价/n', '怎么样/r']
查询结果:  旦苑的评分是: 4.2~
```

问题 5-ut 好吃的菜，（某个校区的味道不错的菜）

问题 5-ut 好吃的菜，（某个校区的味道不错的菜）

```
result = que.process_question("北区好吃的菜")
print(que.question_posseg())
print("查询结果: ",result)
```

```
抽象问题为: ut好吃的菜
使用模版编号：  5
问题模版： ut 好吃的菜
红豆薏米粥 0.7752383512260418
肠粉 0.6573740846802799
糯米肉丸 0.6510968526273782
豌杂面 0.5347966163779838
['北区/ut', '好吃/v', '的/uj', '菜/n']
查询结果:  北区评价不错的菜有：红豆薏米粥、肠粉、糯米肉丸、豌杂面等~
```

问题 6-rr 好吃的菜，（某个食堂的味道不错的菜）

问题 6-rr 好吃的菜，（某个食堂的味道不错的菜）

```
result = que.process_question("南食有什么好吃的")
print(que.question_posseg())
print("查询结果: ",result)
```

```
抽象问题为: rr有什么好吃的
使用模版编号：  6
问题模版： rr 好吃的菜
麻辣香锅 0.6884623355154181
烧腊饭 0.5192396200123331
牛肉木桶饭 0.9984039460685165
糖醋排条 0.7930286318012278
红烧肉 0.9886967751614043
菠萝古老肉 0.9828983215110937
瓦罐汤 0.9942431554546138
['南食/rr', '有/v', '什么/r', '好吃/v', '的/uj']
查询结果:  南食评价不错的菜有：麻辣香锅、烧腊饭、牛肉木桶饭、糖醋排条、红烧肉、菠萝古老肉、瓦罐汤等~
```

问题 7-sst 菜，（某菜系有什么菜）

问题 7-sst 菜，（某菜系有什么菜）

```
result = que.process_question("日式料理有什么")
print(que.question_posseg())
print("查询结果: ",result)
```

```
抽象问题为: sst有什么
使用模版编号：  7
问题模版： sst 菜
['日式料理/sst', '有/v', '什么/r']
查询结果:  日式料理: | 日式猪排饭在旦苑可以吃到 | 蛋包饭在南小食可以吃到 | 铁板饭在南苑可以吃到
```

对于给定的八类问题，本系统基本上可以正确识别并从数据库中查询答案返回。其中第5、6 类问题涉及了情感分类，菜名后面的数字表示了对该菜品评价的情感得分，把味道不错的菜中把阈值设为 0.5，并返回满足条件的菜名。

6 结论和感想

这个系统后续还可以加入很多内容，比如在做成手机 app 或微信小程序的形式以后，可以增加让同学添加评价到数据库中的功能，同时增加图片评论的功能，这样让同学对于复旦食堂的评价不仅仅是靠口耳相传，毕竟真正到大众点评这样的网站上去评价复旦食堂的同学很少，从网站上的评论数也可以看出这一点，而且很少会有同学在去食堂吃饭的时候还特意打开大众点评看评论的，但是如果有了这个复旦美食问答系统，同学在去食堂的路上或者排队的时候可以很方便地获取菜品的评价信息。

此外，现在复旦食堂的承包商和学生之间的沟通途径很少，同学的意见不能很好地反馈给食堂，只有在留言板上能偶尔看到同学留言，但是很多同学即使有一些意见，也不会写在留言板上，但是如果这个系统后续能进一步完善，食堂就可以通过这个系统来得知同学们对各个菜品的评价，从而做出改良，我想这样的话食堂的生意一定会更好，而且同学们也会更加愿意去食堂就餐。与此同时，还可以和食堂合作，增加食堂端，食堂可以对自己的新菜品进行推送，比如学生可以在系统中询问 xx 食堂有什么新品，来知道新的菜式，食堂也可以提供更加详细的菜品的介绍，这样也可以避免食堂的有些新的菜式没有办法很好宣传的情况。

开始做这个 pj 只是为了课程的要求，但是做下来以后觉得这个项目如果可以进一步推进和完善的话是可以给同学们的就餐提供很多的便利，也可以让很多隐藏的美食被更多同学熟知。

这个项目的完成让我对问答系统的建立有了一定的认识，从分词到问题的匹配，实体抽取和数据库查询等，对这些技术都有了一定的掌握。这个系统使用了两个分类的模型，分别是基于朴素贝叶斯的问题分类和 SnowNLP 的文本情感分类，让我对文本分类任务的处理过程和方法有了进一步的了解。之前通过课堂的学习和期末的复习只是了解了很多模型的理论基础，没有具体运用到实际中去，通过这次的 pj 可以更好地把理论和实践结合起来，如果没有之前课堂上学到的理论知识，只知道拿模型来套的话，就不知道如何发挥模型的最大效果，所以理论和实践两者都很重要。

本项目还存在着缺陷有待进一步的完善，首先目前使用的数据规模比较小，只包含了复旦食堂菜品的一部分，而且提供的问题还不够全面，此外，如果要真正运用到实际中去，训练的识别问题的分类器和情感分析器的准确度还有待进一步提高，需要进行更大的训练集，同时可以考虑运用深度学习的模型来提高准确率，本项目中主要是考虑到硬件和查询速度的问题所以用了传统的方法，而且数据集比较小，不太适合深度学习的模型，如果可以扩大数据集再加上硬件的支持的话，就可以得到更加准确的模型。本系统的容错能力也还有待进一步提高，比如如果输入的过程中如果出现错别字的话就不能返回正确的答案。

7 附录

7.1 项目文件列表

- data——数据库数据
  - dish.csv
  - restaurant.csv
- classify.py——问题分类器
- questions.py——问题分类数据
- queTemplate.py——各类问题的模板和查询方法
- queProcess.py——问题的处理同时返回结果
- dict.txt——自定义的词典，词、词频和对应的词性
- test.ipynb——结果演示
- wordcloud——菜品评价的正负样本的词云
- dish_review——菜品评价相关文件
  - review.csv 菜品评价数据
  - sentimentTrain.py 训练情感分类器
  - neg.txt 负样本
  - pos.txt 正样本
- consql——与数据库连接进行查询

7.2 系统运行的方法

本系统运行需要的依赖有：

- python3
- jieba
- sklearn
- snownlp
- pymysql

还需要把 data 文件夹下的数据导入 mysql 数据库中。导入数据库之后，先训练情感分类器，运行 sentimentTrain.py，然后再运行 queProcess.py。