哈尔滨工业大学

<<信息检索>>

实验报告

(2023 年度春季学期)

姓名：	李浩桢
学号：	7203610321
学院：	计算机学院
教师：	张宇

实验二 问答系统设计与实验报告

一、实验目的

本次实验目的是对问答系统的设计与实现过程有一个全面的了解。实验主要内容包括：对给定的文本集合进行处理、建立索引；找出问题的候选答案句并排序；答案抽取，逐步调优。

二、实验内容

1. 文本集合进行处理、建立索引
2. 问题分类
3. 候选答案句排序
4. 答案抽取

三、实验过程及结果

1. 文本集合的处理和索引的建立

利用 Python 自带的数据结构建立了倒排索引（记录词出现的文档编号，在文档中出现的次数），并支持了单个单词的查询以及“与”（&&）模式查询和“或”（||）模式查询。在建立索引前对文本做了分词处理和去停用词处理。索引建立和文档查询集成在 DocSearch 类中，同时 DocSearch 类中还实现了基于 BM25 的文档查询方法，BM25 模型由自己手写实现（BM25.py），根据 corpus 计算每个词的 tf-idf，接着对应每个去除了停用词的查询计算 RSV 值，取 k1=1.5，k3=1.2，b=0.75。再将各文档按 RSV 值排序即可，靠前的文档即为有可能的相关文档。

2. 问题分类

利用 SVM 进行特征学习和训练。核函数取高斯分布核函数，C 取 100，gamma 取 0.01，各超参数由网格搜索得到。特征取 question 中每个词的 tf-idf 值，最终在验证集上得到 0.7825 的准确率。问题分类模型实现在 Question_classifier 类中，类初始化时提供两种模式（train 或 predict），train 模式下可以进行微调超参重新

训练模型并保存，predict 模式下则会直接读取保存路径下的模型并预测问题类型。

3. 候选答案句排序

使用 Ranking SVM 方法实现。选取的特征有：1.候选句的长度；2.问题句和候选句的长度差；3.问题句和候选句的共现单词数；4.问题句和候选句的共现字符数；5.词频向量的余弦相似度；6.tf-idf 向量的余弦相似度；7.候选句与问句之间的 BM25-RSV 值；8.候选句中是否有冒号；9.问题句和候选句的编辑距离。

```
def build_feature(self, q, sent, cv, tv, bm25_score):
    feature_list = []
    q_word, sent_word = q.split(), sent.split()
    tvector_q = tv.transform([q]).toarray().reshape(-1)
    tvector_sent = tv.transform([sent]).toarray().reshape(-1)
    cvector_q = cv.transform([q]).toarray().reshape(-1)
    cvector_sent = cv.transform([sent]).toarray().reshape(-1)
    norm_c = norm(cvector_q) * norm(cvector_sent)
    norm_t = norm(tvector_q) * norm(tvector_sent)

    feature_list.append(f'1:{len(sent_word)}')
    feature_list.append(f'2:{abs(len(q) - len(sent))}')
    feature_list.append(f'3:{len(set(q_word) & set(sent_word))}')
    feature_list.append(f'4:{len(set(q) & set(sent))}')
    feature_list.append(f'5:{np.dot(cvector_q, cvector_sent) / norm_c if norm_c else 0}')
    feature_list.append(f'6:{np.dot(tvector_q, tvector_sent) / norm_t if norm_t else 0}')
    feature_list.append(f'7:{bm25_score}')
    a = 1 if ':' in sent or '：' in sent else 0
    feature_list.append(f'8:{a}')
    query, sentence = q.replace(' ', ''), sent.replace(' ', '')
    feature_list.append(f'9:{Levenshtein.distance(query, sentence)}')
    return feature_list
```

经测试，验证集（取百分之 10 的训练集）上 perfect match 率（第一个句子即答案句）为 0.60 左右，MRR 值为 0.74 左右。由于 RankingSVM 的输出文件可读性太差，在 evaluate 方法中实现了将 RankingSVM 的输出文件转换为可读性好的 json 文件（将候选句按可能性排序好放入‘sentence_chosen_by_model’项中），也方便我们计算 MRR。同时在四模块联合使用时尝试实现了将 BM25 检索出的前三个文档的所有句子都作为候选句进行排序，但经过答案抽取后效果不佳，故作罢。

4. 答案抽取

根据问题分类的类型选择不同的规则进行抽取。考虑到答案不一定一定就在候选句列表首位，故我们考虑排名靠前的 6 个候选句，并依次进行答案抽取，若未抽取到答案则进入下一个候选句，否则结束抽取。由于问题分类对大类的分类效果良好，故答案抽取时先考虑大类，对于‘HUM’类问题，我们抽取候选句中的人名，对于‘LOC’类问题则抽取地名，对‘NUM’类则抽取数

词，对‘DES’类则选取第一句作为答案。同时特殊处理带冒号的候选句以及类型为‘TIME’的问题对应的候选句。对于‘TIME’类则设置特定正则表达式规则，进行匹配得到最终答案。仅考察抽取单模块，BLEU1 值为 0.47 左右。

四、实验心得

四模块串联在整个 train 上得到最终结果的 BLEU1 值约为 0.25。整体来说实验并不难，管理好各模块的接口和模型、数据路径即可事半功倍。第三部分 RankingSVM 输入数据输出数据处理较为麻烦，需要细心处理。大部分时间花在调整超参，调整输入特征和调整抽取规则上，需要在自己划分出的验证集上测试并不断优化自己的模型和规则。