哈尔滨工业大学

<<信息检索>>

实验报告

(2022 年度春季学期)

姓名:	王艺丹
学号:	1190201303
学院:	计算机学院
教师:	张宇

实验二 问答系统设计与实现

一、实验目的

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

(一)文本集合进行处理、建立索引

对所有 passage_multi_sentences.json 文档分词、分句并建立索引，作为问答系统的检索语料。（自己实现的检索系统有加分）

(二)问题分类

使用**机器学习**的方法训练一个问题分类模型，得到问题类别信息，然后将其融入到候选答。

(三)候选答案句排序

使用**机器学习**的方法对所有候选答案句按照其包含正确的可能性进行排序，可能性越大的越靠前。

(四)答案抽取

基于规则的方法或者使用机器学习，完成从候选答案句中抽取精简的答案片段。

三、实验过程及结果

综述：

utils.py：实现各功能函数复用

- get_stop_dic(file_path)：构建停用词字典树
- tokenize(content:str)：分词及去停用词处理
- get_pos(sent)：对 sent 进行分词处理，并利用 LTP 库进行词性标注，返回分词列表与词性列表
- get_ner(sent)：对 sent 进行分词处理，并利用 LTP 库进行命名体识别，返回实词数

BM25.py：类 BM25，自己实现 bm25 检索模型

- cal_scores(self, query)：实现对已分词处理的 query 进行 bm25 得分的计算，利用全 0 向量直接对全部文档进行计算
- get_topk(self, query, k=3)：实现对已分词处理的 query 进行计算检索，返回 bm25 得分最高的 k 个文档对应的索引

lr.py：类 LR_b 与 LR_s，实现对大类小类的 LR 分类模型

闪光点：

✧ 自己实现 BM25 检索模型：

见 bm25.py；

且利用全 0 向量实现对全文档的 bm25 得分直接计算，空间换时间提高效率

```
def cal_scores(self, query): # 利用全0向量直接计算全部索引文档
      scores = np.zeros(self.corpus_size)
      doc_len = np.array(self.doc_len)
      for q in query.split(' '): # 分词处理好的query
            q_freq = np.array([(doc.get(q) or 0) for doc in self.doc_freqs])
            scores += (self.idf.get(q) or 0) * (q_freq * (self.k1 + 1) /
                                                (q_freq + self.k1 * (1 - self.b + self.b * doc_len / self.avgdl)))
      return scores

def get_topk(self, query, k=3):
      scores = self.cal_scores(query)
      topk = np.argsort(scores)[::-1][:k] # 返回对应的索引
      return [self.index[i] for i in topk]
```

2.1 构建检索

实现 BM25 模型，实现 tf-idf 等特征的计算，直接利用文档对应的 pid 作为索引，代码如下：

```
1. import math
2. import numpy as np
3. from tqdm import trange, tqdm
4.
5.
6. class BM25():
7.     def __init__(self, idx, corpus, k1=1.5, b=0.75, epsilon
   =0.25):
8.         self.index = idx
9.         self.corpus = corpus
10.        self.k1 = k1
11.        self.b = b
12.        self.epsilon = epsilon
13.        self.corpus_size = len(self.corpus)
14.        self.avgdl = 0
15.        self.doc_freqs = []
16.        self.idf = {}
17.        self.doc_len = []
18.        nd = self._initialize()
19.        self._calc_idf(nd)
20.
21.    def _initialize(self):
22.        nd = {}  # numdoc: word -> number of documents with
    word
23.        num_doc = 0
```

```
24.         for document in self.corpus:
25.             document = document.split(' ')
26.             self.doc_len.append(len(document))  # 每篇doc的词的数量
27.             num_doc += len(document)    # 全部词的总数
28.
29.             freq = {}
30.             for word in document:
31.                 if word not in freq:
32.                     freq[word] = 0
33.                 freq[word] += 1
34.             self.doc_freqs.append(freq)
35.
36.             for word, f in freq.items():
37.                 nd[word] = nd.get(word, 0) + 1 #拉普拉斯平滑
38.#                   try:
39.#                       nd[word]+=1
40.#                   except KeyError:
41.#                       nd[word] = 1
42.
43.         self.avgdl = num_doc / self.corpus_size #文档平均长度
44.         return nd
45.
46.     def _calc_idf(self, nd):
47.         idf_sum = 0
48.         negative_idfs = []
49.         for word, freq in nd.items():
50.             idf = math.log(self.corpus_size - freq + 0.5) - math.log(freq + 0.5)
51.             self.idf[word] = idf
52.             idf_sum += idf
53.             if idf < 0:
54.                 negative_idfs.append(word)
55.         self.average_idf = idf_sum / len(self.idf)
56.
57.         eps = self.epsilon * self.average_idf
58.         for word in negative_idfs:
59.             self.idf[word] = eps
60.
61.     def cal_scores(self, query): # 利用全0向量直接计算全部索引文档
62.         scores = np.zeros(self.corpus_size)
63.         doc_len = np.array(self.doc_len)
```

```
64.         for q in query.split(' '): # 分词处理好的query
65.             q_freq = np.array([(doc.get(q) or 0) for doc in
    self.doc_freqs])
66.             scores += (self.idf.get(q) or 0) * (q_freq * (s
   elf.k1 + 1) /
67.                                                 (q_freq + se
   lf.k1 * (1 - self.b + self.b * doc_len / self.avgdl)))
68.         return scores
69.
70.     def get_topk(self, query, k=3):
71.         scores = self.cal_scores(query)
72.         topk = np.argsort(scores)[::-1][:k] # 返回对应的索引
73.         return [self.index[i] for i in topk]
```

3.2 问题分类

最初采用 LighGBM 机器学习模型进行分类，大类准确率仅有 0.72：

In [86]:
```
#lightgbm
print(classification_report(df_test['Blabel'], y_pre))
```

	precision	recall	f1-score	support
0	0.94	0.68	0.79	178
1	0.78	0.72	0.75	390
2	0.84	0.95	0.89	244
3	0.79	0.67	0.72	153
4	0.48	0.56	0.52	194
5	0.51	0.66	0.58	153
6	0.00	0.00	0.00	3
accuracy			0.72	1315
macro avg	0.62	0.60	0.61	1315
weighted avg	0.74	0.72	0.72	1315

后考虑 LR 分类器训练类别数量个分类器，首先对句子进行分词以及去停用词处理，利用 TfidfVectorizer 进行特征提取，对大类已经小类分别构建一个逻辑回归分类模型并利用网格搜索进行参数优化，最终**大类准确率达到 0.90**：

]:
```
# precision:0.91
y_pre_s = lr.predict(df_test)
print(classification_report(df_test['Blabel'], y_pre_s))
```

	precision	recall	f1-score	support
DES	0.90	0.86	0.88	153
HUM	0.95	0.88	0.91	178
LOC	0.94	0.88	0.91	390
NUM	0.98	0.97	0.97	244
OBJ	0.73	0.92	0.81	194
TIME	0.95	0.91	0.93	153
UNKNOWN	1.00	0.33	0.50	3
accuracy			0.90	1315
macro avg	0.92	0.82	0.84	1315
weighted avg	0.91	0.90	0.90	1315

小类准确率达到 0.78：

TIME_ERA	1.00	0.64	0.78	25
TIME_HOLIDAY	0.00	0.00	0.00	1
TIME_MONTH	0.75	1.00	0.86	3
TIME_OTHER	0.93	0.90	0.91	41
TIME_RANGE	0.29	0.50	0.36	4
TIME_SEASON	1.00	1.00	1.00	4
TIME_SOLARTERM	1.00	0.75	0.86	4
TIME_WEEK	1.00	1.00	1.00	4
TIME_YEAR	0.88	0.92	0.90	49
UNKNOWN_	1.00	0.67	0.80	3
accuracy			0.78	1315
macro avg	0.70	0.72	0.68	1315
weighted avg	0.82	0.78	0.78	1315

代码如下：

```
# 提取正文，没有则用标题充当正文内容
content = soup.find_all('p')
if content is None:
    paragraphs = title
else:
    paragraphs = ''
    for i in range(len(content)):
        paragraphs = paragraphs + content[i].get_text()
```

3.3 候选答案句排序

由于该部分必须使用机器学习方法，上网搜索最终选取 RankingSVM 模型进行训练预测，特征构建，模型训练预测方法参考官方网站

最初选取 bm25 相似度、tfidf 相似度、词向量相似度、余弦相似度、实词数、LCS 长度占比、nigram 与 bigram 词共现比例、编辑距离 9 个特征用于模型训练，后续对特征优化进行对照实验发现词向量相似度与余弦相似度会造成模型准确率下降，故最终选取其余 7 个特征用于模型训练，各特征计算函数的具体实现不做赘述，见 answer_sentence_selection.py

统计处理发现各文章平均句子数量为 20，于是对训练数据中相关的句子相关度赋值为 20，其余赋值为 0

部分特征数据展示如下：

<<信息检索>>实验报告

```
0 qid:4695 1:2.143177259828395 2:0 3:26 4:22 5:0.16129032258064516 6:0.12903225806451613 7:0.5555555555555556
0 qid:4695 1:3.706815561729829 2:0 3:7 4:5 5:0.45454545454545453 6:0.36363636363636365 7:0.5555555555555556
20 qid:4695 1:1.792401618551133 2:0 3:9 4:5 5:0.125 6:0.0 7:0.125
0 qid:4695 1:0.2839369742596776 2:0 3:10 4:5 5:0.1 6:0.0 7:0.1111111111111111
0 qid:4695 1:0.0 2:0 3:9 4:8 5:0.0 6:0.0 7:0.0
0 qid:4695 1:0.0 2:0 3:19 4:15 5:0.0 6:0.0 7:0.0
0 qid:4695 1:0.0 2:0 3:9 4:7 5:0.0 6:0.0 7:0.0
0 qid:4696 1:1.8345924390440518 2:0 3:10 4:8 5:0.09090909090909091 6:0.0 7:0.16666666666666666
0 qid:4696 1:0.9050245389300962 2:0 3:16 4:6 5:0.058823529411764705 6:0.0 7:0.16666666666666666
20 qid:4696 1:1.0891840306012226 2:0 3:11 4:3 5:0.09090909090909091 6:0.0 7:0.16666666666666666
0 qid:4696 1:0.0 2:0 3:7 4:2 5:0.0 6:0.0 7:0.0
0 qid:4696 1:0.0 2:0 3:7 4:2 5:0.0 6:0.0 7:0.0
```

模型训练与预测：

```python
# 调用svm-rank可执行文件，训练并预测模型
def train_rank_svm(train_data_path, model_path):
    train_cmd = f'.\svm_rank\svm_rank_learn.exe -c 200.0 {train_data_path} {model_path}'
    os.system(train_cmd)

def test_rank_svm(test_data_path, model_path, pre_path):
    predict_cmd = f'.\svm_rank\svm_rank_classify.exe {test_data_path} {model_path} {pre_path}'
    os.system(predict_cmd)
```

最终模型精度如下：

```python
# LCS 200 7 20-0
# test_rank_svm(test_data_path, model_path, pre_path)
idxs = get_ans(test_data_path, pre_path)
```

```
测试集共计qid:1063个 正确数:670个 模型准确率:0.6302916274694261
```

3.4 答案抽取

该部分采用**基于规则**的方法，首先对候选句排名第一的句子进行分词去停用词处理，在进行分类预测大类小类对应的类别，最终基于规则匹配进行答案抽取。

各类规则如下：

考虑一个 trick：一般答案句中出现冒号时，冒号后往往对应的是正确的答案句，所以各类均先进行判断有无冒号，若有，则返回冒号后对应的原句子：

```python
# 对冒号的处理函数
def drop_mh(sent):
    if '：' in sent or ':' in sent or '：' in sent:
        begin = sent.index('：') if '：' in sent else sent.index(':')  # 冒号
        aa = sent[begin+1:]
    else:
        aa = sent
    return aa.strip()
```

(1) HUM：

该类问题答案平均 BLEU1 值为 0.55

进行分类以及词性标注，若小类别为人物，则返回顿号连接的 Nh 对应的人名；若小类别为机构，则返回顿号连接的 Ni 对应的机构名以及地名；若句子

中含有冒号，则返回冒号后的句子

```
# HUM 0.5559143590965517
def get_hum_ans(qa_list):
    ans = []
    qids = []
    qlst = []
    for qa in tqdm(qa_list):
        ques = qa['q']
        sent = qa['a']
        qlst.append(ques)
        qids.append(qa['qid'])
        lb, ls = qa['labels'].split('_') #大小标签
        if lb != 'HUM':
            ans.append(drop_mh(sent))
            continue
        ners = get_ner(sent)
        if ners['Nh']:
            ans.append('、'.join(ners['Nh']))
        elif ls == 'ORGANIZATION'and ners['Ni']:
            ans.append('、'.join(ners['Ni']))
        elif '：' in sent or ':' in sent:
            ans.append(ex_mh(sent))
        else:
            ans.append(drop_mh(sent))
    return pd.DataFrame({'qid':qids, 'question':qlst, 'answer':ans})
```

(2) NUM:

该类问题答案平均 BLEU1 值为 0.59

进行词性标注，若数词后紧跟着的为量词，则将二者合并，返回数词列表中的第一个数词。

```
# NUM 0.5926360860213366
def get_num_ans(qa_list):
    ans = []
    qids = []
    qlst = []
    for qa in tqdm(qa_list):
        ques = qa['q']
        sent = qa['a']
        qlst.append(ques)
        qids.append(qa['qid'])
        if '：' in sent or ':' in sent:
            ans.append(ex_mh(sent))
            continue
        seg, pos = get_pos(sent)
        result = []
        for idx, tag in enumerate(pos):
            if tag == 'm' and idx < len(pos) - 1:
                if pos[idx + 1] == 'q':
                    result.append(seg[idx] + seg[idx + 1])
                else:
                    result.append(seg[idx])
        if len(result) > 0:
            ans.append(result[0])
        else:
            ans.append(drop_mh(sent))
    return pd.DataFrame({'qid':qids, 'question':qlst, 'answer':ans})
```

(3) LOC：

该类问题答案平均 BLEU1 值为 0.49

进行词性标注，返回顿号连接的 Ni 词性对应的词

```python
# # LOC 0.48676425856487476
def get_loc_ans(qa_list):
    ans = []
    qids = []
    qlst = []
    for qa in tqdm(qa_list):
        ques = qa['q']
        sent = qa['a']
        qlst.append(ques)
        qids.append(qa['qid'])
        if '：' in sent or ':' in sent:
            ans.append(ex_mh(sent))
            continue
        ners = get_ner(sent)
        if ners['Ns']:
            ans.append('、'.join(ners['Ns']))
            continue
        else:
            ans.append(drop_mh(sent))
            continue
    return pd.DataFrame({'qid':qids, 'question':qlst, 'answer':ans})
```

(4) TIME

该类问题答案平均 BLEU1 值为 0.65

基于正则匹配，具体规则思想见下图：

```python
# TIME 0.645968083728072
def get_time_ans(qa_lst):
    ans = []
    qids = []
    qlst = []
    for qa in tqdm(qa_lst):
        ques = qa['q']
        sent = qa['a']
        qlst.append(ques)
        qids.append(qa['qid'])
        lb, ls = qa['labels'].split('_') # 大小标签
        result = []
        if ls == 'YEAR': # xx年/xxxx年
            result = re.findall(r'\d{2,4}年', sent)
        elif ls == 'MONTH': # x月/xx月
            result = re.findall(r'\d{1,2}月', sent)
        elif ls == 'DAY': # x日/xx日
            result = re.findall(r'\d{1,2}日}', sent)
        elif ls == 'WEEK':
            result = re.findall(r'((周|星期|礼拜)[1-7一二三四五六日])', sent)
            result = [res[0] for res in result]
        elif ls == 'RANGE': # xxxx年到xxxx年/xxx年-xxxx年
            result = re.findall(r'\d{2,4}[年]?[-到至]\d{2,4}[年]?', sent)
        else:
            result = re.findall(r'\d{1,4}[年/-]\d{1,2}[月/-]\d{1,2}[日号]?', sent)  # 年月日
            if not result:
                result = re.findall(r'\d{1,4}[年/-]\d{1,2}月?', sent)  # 年月
            if not result:
                result = re.findall(r'\d{1,2}[月/-]\d{1,2}[日号]?', sent)  # 月日
            if not result:
                result = re.findall(r'\d{2,4}年', sent)
            if not result:
                result = re.findall(r'\d{1,2}月', sent)

        if len(result) > 0:
            ans.append(result[0]) # 返回结果中的第一个日期
        else:
            ans.append(drop_mh(sent))
    return pd.DataFrame({'qid':qids, 'question':qlst, 'answer':ans})
```

(5) DES/UNKOWN

该类问题答案平均 BLEU1 值为 0.54

无明显规则，仅利用 trick 进行冒号后的答案提取

```
# DES/UNK 0.5433658533737482
# DES:0.5433658533737482
# UNK:0个样例
def get_do_ans(qa_lst):
    ans = []
    qids = []
    qlst = []
    for qa in tqdm(qa_lst):
        ques = qa['q']
        sent = qa['a']
        qlst.append(ques)
        qids.append(qa['qid'])
        if ': ' in sent or ':' in sent:
            ans.append(ex_mh(sent))
        else:
            ans.append(drop_mh(sent))
    return pd.DataFrame({'qid':qids, 'question':qlst, 'answer':ans})
```

最终结果：
利用训练数据进行验证，平均 BLEU1 值为 0.55

In [26]:
```
# 0.5512000673111742
cal_bleu(all_ans['answer'], real_ans['ans'])
```

```
G:\Anaconda3\lib\site-packages\nltk\translate\bleu_score.py:516: UserWarning:
The hypothesis contains 0 counts of 4-gram overlaps.
Therefore the BLEU score evaluates to 0, independently of
how many N-gram overlaps of lower order it contains.
Consider using lower n-gram order or use SmoothingFunction()
  warnings.warn(_msg)
G:\Anaconda3\lib\site-packages\nltk\translate\bleu_score.py:516: UserWarning:
The hypothesis contains 0 counts of 3-gram overlaps.
Therefore the BLEU score evaluates to 0, independently of
how many N-gram overlaps of lower order it contains.
Consider using lower n-gram order or use SmoothingFunction()
  warnings.warn(_msg)
G:\Anaconda3\lib\site-packages\nltk\translate\bleu_score.py:516: UserWarning:
The hypothesis contains 0 counts of 2-gram overlaps.
Therefore the BLEU score evaluates to 0, independently of
how many N-gram overlaps of lower order it contains.
Consider using lower n-gram order or use SmoothingFunction()
  warnings.warn(_msg)
```

```
共计5352个样例，平均bleu为:0.5512000673111742
```

Out[26]:
```
0.5512000673111742
```

<<信息检索>>实验报告

实验结果：


```
{"qid": 0, "question": "北京依道奥企业管理咨询有限公司致力于为哪些行业提供高级人才？", "answer_pid": [10501, 8301, 7684], "answer": "北京依道奥致力于为房地产、金融、ICT、医药、医疗器械、消费品、能源、矿产行业提供高级人才配置解决方案；同时，为非特定行业背景的人力资源专业岗位提供配置解决方案。"}
{"qid": 4, "question": "安康文庙的二期工程是什么时候开始的？", "answer_pid": [10504, 6854, 2835], "answer": "2010年10月"}
{"qid": 12, "question": "澳城的售楼处在什么位置？", "answer_pid": [7157, 8112, 5981], "answer": "暂无资料"}
{"qid": 32, "question": "焊接设备与工艺的封面是谁设计的？", "answer_pid": [10532, 13809, 987], "answer": "焊接设备与工艺[1]"}
{"qid": 40, "question": "艾华士29324（10.4英寸笔记本包/枣红）有多大？", "answer_pid": [13274, 8179, 4628], "answer": "台式机箱（中塔）"}
{"qid": 62, "question": "物业管理品牌化的发展过程中，1988年发生了什么？", "answer_pid": [10562, 1900, 5289], "answer": "1981年到1994年期间，深圳物业管理的发展过程中有几笔值得特别提及。"}
{"qid": 71, "question": "赤狐-041每小时可以跑多少公里？", "answer_pid": [10571, 6102, 11320], "answer": "50-90厘米"}
{"qid": 75, "question": "中国（桂林）运通国旅国际会议奖励中心做过哪个公司的经销商会议？", "answer_pid": [10575, 1769, 4955], "answer": "中国（、桂林）、运通国旅国际会议、中心、运通国旅会奖、旅行、中国桂林运通国际旅行社、有限公司"}
{"qid": 112, "question": "贵州省企业职工基本养老金计发办法国发号是多少？", "answer_pid": [10612, 13556, 9250], "answer": "根据《国务院关于完善企业职工基本养老保险制度的决定》（国发200538号）和《省人民政府关于完善企业职工基本养老保险制度的通知》（黔府发200621号）的规定，结合我省实际，制定本办法。"}
{"qid": 115, "question": "特优1102植株有多高？", "answer_pid": [10615, 8368, 11365], "answer": "株型适中，熟期转色好，结实率高，每亩有效穗数17.7万，株高116.2厘米，穗长23.5厘米，每穗总粒数120.4粒，结实率87.4%，千粒重30.7克"}
{"qid": 119, "question": "明文伯仁西洞庭山图的作者在中年以后曾经去过哪些地方？", "answer_pid": [10619, 9202, 14565], "answer": "北京、南京、松江"}
{"qid": 133, "question": "临时停车规定驾驶员应该怎么做？", "answer_pid": [10634, 12910, 7898], "answer": "一般人不离车都可以停车等候，驾驶员不能离开车内。"}
{"qid": 146, "question": "59-30-3加热至多少度可以变色？", "answer_pid": [9668, 12722, 1591], "answer": "40-100度"}
{"qid": 151, "question": "西瓜锈病在哪些月份发病严重？", "answer_pid": [10652, 12900, 9699], "answer": "发病严重叶片早期枯死。"}
{"qid": 158, "question": "愉快的秘密的作者是谁？", "answer_pid": [10659, 6136, 6377], "answer": "芳香疗法、运动疗法、还有成人游戏，这些男人们贪婪地享受身体上的快感，不管是谁带来的激情都好，谈恋爱、嘿咻，总是让人这么愉快，这些全都是男性们的秘密生活哟。"}
{"qid": 164, "question": "珠溪镇地质灾害防治安全工作管理办法出台的目的是什么？", "answer_pid": [10665, 10855, 9746], "answer": "《珠溪镇地质灾害防治安全工作管理办法》是为了切实搞好地质灾害防治工作，保护人民生命财产安全，保护我镇经济社会持续稳定发展，经镇政府研究特制定的管理办法。"}
{"qid": 192, "question": "严稚棱的英文名叫什么？", "answer_pid": [10694, 4960, 7082], "answer": "严稚棱出生于2010年2月6日早上六时[3]?，英文名Cayla。"}
{"qid": 193, "question": "19世纪英国海军指挥剑(19世纪英国海军指挥剑之一)现在被收藏在哪里？", "answer_pid": [769, 1275, 10695], "answer": "英国"}
{"qid": 198, "question": "微英语传递的是什么精神？", "answer_pid": [10700, 5282, 9138], "answer": "最主要的原因它传递了一种“水滴石穿“的精神，中国历来强调”团结就是力量“，也许一个人的能力微乎其微，但是众志成城产生的力量会撼动世界。"}
{"qid": 203, "question": "四川华信富恒科技有限公司提供哪些增值服务？", "answer_pid": [10705, 9773, 14132], "answer": "提供IBM、HP/ALPHA、SUN 小型机及ORACLE、SYSBASE、DB2数据库保修、升级、租赁、培训等增值服务。"}
```


实验结果文件

实验中的问题：

在使用 TfidfVectorizer 进行特征提取时，发现词汇表有误，查阅资料得知 token_pattern 这个参数默认只匹配长度≥2 的单词，因为长度为 1 的单词在英文中一般是无足轻重的，所以在进行中文处理时，改为：

TfidfVectorizer(token_pattern=r"(?u)\b\w+\b")

四、实验心得

- 学会构建 RankingSVM 的训练特征
- 学会训练使用 RankingSVM 模型
- 对 frep、tf、tf-idf 等特征有了更为深刻的认识
- 掌握了 BM25 算法
- 对问答系统的简易框架有了认识与实践