建立小型搜索引擎实验报告

关开思
2020201411

2021 年 7 月 5 日

1 整体介绍

本项目总工分为六天完成。在本次编程集训中针对以下五个网站：

- 中国人民大学教务处（‘http://jiaowu.ruc.edu.cn’）
- 中国人民大学教师发展中心 (‘http://ctd.ruc.edu.cn’)
- 中国人民大学访学网 (‘http://fangxue.ruc.edu.cn’)
- 中国人民大学国际小学期官网 (‘http://iss.ruc.edu.cn’)
- 中国人民大学学生处官网 (‘http://xsc.ruc.edu.cn ‘)

并且使用爬虫，爬取了 9536 个网页，之后使用 Beautifulsoup 包提取相关的正文以及标题进行保存，并在此基础上进行分词，构建倒排索引字典并保存。在第三天时实现了布尔值 and 和 or 的搜索结果，之后构建向量空间模型，将每个文档以及搜索内容都视为一个向量，使用 tfidf 加权计算得分，之后进行计算余弦相似度，并根据余弦相似度排顺序，返回结果最大的前 20 位。在最后一天，将自己之前的模型，导入到 webUI 的模型框架之中，并用 CSS 工具对搜索引擎前端界面进行了简单的美化，完成项目结果。

2 实现流程

我实现的扩展功能是实现了爬虫终端后继续，tf-idf 加权计算，以及 web 网页的简单优化处理

2.1 爬虫的实现

爬虫主要采用 requets 包进行模拟发送 get 请求，提取网站的页面的全部代码，并且以 html 文件的形式保存在本地电脑之中，并使用 BeautifulSoup 对爬取下来的网页源代码进行解析，提取网页的标题以及正文并分别以 txt 文档的形式保存。

其中较为关键的一点是，我们在 requests.get() 爬取网页源代码时，要保存网页所包含的全部链接，并将这些链接作为之后爬虫的目标。所以这里的爬虫要**涉及到 BFS（广度优先搜索）的思想，利用数据结构队列（在 python 之中列表可以当作队列使用）从一直的 "种子" URLs 开始，抓取并解析它们，同时将抓取到的 URLs 加入到队列之中，循环进行**。

这样我们就可以实现，以这五个种子网页为根基，相似度由高到低的排列，在爬取 9000+ 网页之后，网页大概已经包括这五个目录下的全部子网页。

同时，另一个比较关键的一点就是**要正确处理所获取的 url，正确处理绝对路径以及相对路径的关系**，我们爬取所获得的 url 包括了相对路径以及绝对路径，同时还包括了一些无关的友情链接，我在第一次爬虫时，就是没有删除友情链接，导致后来爬取大量的于种子网站无关的网页。所以要注意两点：删除前面域名不是五个种子 url 的链接；对于相对路径，我们要在前面加上相对应的种子 url 使其，成为能够直接访问的链接。（关键代码在第三部分中体现）

2.2 倒排索引的构建

首先，在上一步骤的基础上，用 Beautifulsoup 提取 html 文件的正文以及标题部分，保存到 txt 文档之中，正文的 txt 文档构成待检索集合。**构建倒排索引，实际上就是要用变长列表来保存词出现的信息**。首先我们要将每个文档进行分词处理，并且依次遍历每一个词，构建（term，docid）对组成序列的列表，之后排序，对于元素为元组列表的排序，首先它会按照第一个元素进行排序，第一个元素相同时，按照第二个元素进行排序。在（term，docid）对构建好之后，我们可以用它来构建倒排索引的字典，其中有一个小技巧就是用 collections 包中的 defaultdict 字典，在键值对不存在时我们就可以自动生成含有元素为-1 的列表。

在进行玩倒排索引之后，我们就可以进行简单的布尔查询，我构建了关于 and 和 or 的布尔查询支持，这里用到了归并的思想方法

2.3 倒排索引的改进以及保存词频和文档信息

对于专业用户而言，布尔查询的结果是有效的，但是对于普通用户，普通用户不会而且也不愿意组织和提交布尔查询，同时也不希望浏览大量的与查询匹配的文档，所以我们要进行排序检索，即与返回一个文档集合的布尔检索模型不同，我们要返回一个有序的结果列表

这里要用到**词频加权以及逆文档频率加权处理**，词频分数计算公式：$w_{t,d} = 1 + log_{10} tf_{t,d}$，从而我们可以据此对多个词的查询进行打分 $Score = \Sigma_{t \in q \cap d}(1 + logtf_{t,d})$, 为了方便记录 docid 以及我们构建了 Posting 类，这样让 index 字典的 value 都是元素为 Posting 类的列表。

接下来一个重要的问题就是如何保存 invertedindex 字典，对于普通字典，我们可以通过建 josn 包来进行保存而对于这种含有特殊元素的字典，pickle 包可以派上用场，pickle.dump 和 pickle.load 可以很方便的保存和载入特殊元素的字典，pickle 包可以将对象序列化，将结果以数据流的方式写入文件对象之中，从而可以保存特殊元素的数据类型而不用进行转化。

2.4 基于余弦相似度计算的排序检索

我们可以构建一个**向量空间模型，将文档以及查询都用向量来表示**这样我们可以通过两个向量之间的距离来表示向量之间的相似度，而因为向量本身的长度会影响欧式距离，所以我们用两向量之间的夹角来表示两个向量之间的相似度。

tf-idf 加权得分计算：$W_{t,d} = (1+log_t ft, d) * log(N/df_t)$, 一个词在文档中出现的次数越多，这个词在待检索系统中越罕见，那么这个词对相关性的贡献就越大。文档的最终得分：$Score = \Sigma_{t \in q \cap d} tf.idf_{t,d}$, 在长度归一化之后，查询向量与文档向量的内积就是夹角值 $cos(\vec{q}, \vec{d}) = \frac{\vec{q}\vec{d}}{|\vec{q}||\vec{d}|} = \frac{\sum_{i=1}^{|V|} q_i d_i}{\sqrt{\sum_{i=1}^{|V|} q_i^2} \sqrt{\sum_{i=1}^{|V|} d_i^2}}$

$cos(\vec{q}, \vec{d}) = \sum_{i=1}^{|V|} q_i d_i$

总而言之，将查询结果和文档表示成 tf-idf 向量，通过计算查询向量和文档向量之间的预先相似度进行评分按照余弦相似度的大小关系进行排序，将 top20 的结果返回给用户。

2.5 WebUI 的实现

将之前自己所写的代码进行封装，接口为输入的字符串 key 而输出为元素值为字典的列表，这一步骤的关键是如何从保存的域名和 title 的 txt 文件之中提取信息，结合之前的返回前 20 位的 url 构建元素为’ title ‘: title,’ url ‘:’ url 的 results 列表，这样就完成了一个简单的搜索引擎。之后就是用 CSS 的知识，对搜索引擎进行一些基本的优化，这样就大体上完成了今天的任务。

WebUI 建立的时候有一个很大的问题就是，无法正常的导入图片，主要原因就在于 Web server 在 python 语言下无法正确的解析绝对路径，所以我们要额外在 main.py 加上参数,static-folder=’static’,static-url-path=’/static’,这样把图片放在 static 文件夹之下，通过解析相对路径就能正常的导入文件

3 代码细节

```
with open('queue', 'wb') as f:
    pickle.dump(queue, f)
    print('队列保存成功')
with open('queue','rb') as f:
    queue = pickle.load(f)
    print("队列载入成功")
```

Listing 1: 实现爬虫终端后继续

```
while len(queue)!=0:
    count +=1
    url = queue.pop(0)
    print(queue)
    used_urlset.add(url)
    html_doc = get_html(url,headers = headers)
    url_sets = crawl_all_urls(url,html_doc)
    print(url_sets)
    ##这一步是将遍历过的节点不在加入到队列之中，加玩之后将队列保存
    try:
        for new_url in url_sets:
            if new_url in used_urlset:
                continue
            else:
                queue.append(new_url)
      ##保存html文件
```

```
        if count < 10:
            path = htmlpath + '0' + str(count) + '.html'
        else:
            path = htmlpath + str(count) + '.html'
        with open(path, 'w', encoding='utf-8') as f:
            print('保存{}'.format(path))
            f.write(html_doc)
            f.close()
        dic[count] = url
        savedic(dic)
        if wait_time > 0:
            print("等待{}秒之后开始抓取".format(wait_time))
        time.sleep(wait_time)
    except:
        print('失败了但是要继续')
        continue
```

Listing 2: 爬虫部分之广度优先搜索

```
###构造term_docid_pairs
term_docid_pairs = []
for docid , filename in enumerate(collections):
    with open(os.path.join(filepath1,filename),encoding='utf-8') as fin :
        for term in jieba.cut(fin.read()):
            ##必要的过滤，建议实现一个更复杂的函数,目的是出去哪些空格，标点符号
            if len(term.strip())<=1:
                continue
            term_docid_pairs.append((term,docid))
term_docid_pairs = sorted(term_docid_pairs)

##为了编程实现方便，我们在每个posting list开头加上了一个用来标记开头的-1，docid从0 开始

inverted_index = defaultdict(lambda :[-1])

for term,docid in term_docid_pairs:
    postings_list = inverted_index[term]
    if docid != postings_list[-1]:
        postings_list.append(docid)
```

Listing 3: 倒排索引的构建

```
collections = [file for file in os.listdir('保存的网页') if os.path.splitext(file)[1] =='.txt']
# collections = collections[:501]

```

```
N = len(collections)
print(N)
#N为集合中文档的数量
##构造term_docid_pairs，元组对列表，还有文件长度列表,因为但是运行的时间过长，我们将term_docid_pairs保存到一个txt文档之中
term_docid_pairs = []
doc_length = []
for docid , filename in enumerate(collections):
    with open(os.path.join(filepath1,filename),'r',encoding='utf-8') as fin:
        terms = [term for term in jieba.cut(fin.read()) if len(term.strip())>1]
        for term in terms:
            term_docid_pairs.append((term,docid))
        print('done',docid)

        ##统计tf，用Counter字典定义了这个term出现的频数，value就是
        term_counts = np.array(list(Counter(terms).values()))

        log_tf = 1 + np.log(term_counts)
        doc_length.append(np.sqrt(np.sum(log_tf**2)))##用词汇频率定义文本长度

with open('lenthgy','wb') as f:
    pickle.dump(doc_length,f)
    print("列表保存成功")
# term_docid_pairs = sorted(term_docid_pairs)
# #构造倒排索引
# #posting list中每一项为Postin类对象
class Posting(object):
    def __init__(self,docid,tf=0):
        self.docid = docid
        self.tf = tf
    def __repr__(self):
        return '<docid:%d, tf: %d>'%(self.docid,self.tf)
#为了编程实现方便，我们在每个posting list开头加上了一个用来标记开头的-1，docid从0 开始
#注意这段代码在只有termdocid派人但是已经排序时才是正确的
#思考一下，如果不用Posting这个类，只用元组的列表是否合适
inverted_index = defaultdict(lambda: [Posting(-1,0)])
##这一步是为了构建Postinglist即变长度列表，列表的元素都是Posting类其中储存了docid和tf信息
for term,docid in term_docid_pairs:
    posting_list = inverted_index[term]
    if docid!=posting_list[-1].docid:
        posting_list.append(Posting(docid,1))
```

```
    else:
        posting_list[-1].tf+=1
inverted_index = dict(inverted_index)##注意了我们需要的是什么
#将这个列表保存
with open('dictionary','wb') as f:
    pickle.dump(inverted_index,f)
    print('字典保存成功')
with open('dictionary','rb') as f:
#     inverted_index = pickle.load(f)
#     print("字典载入成功")
#
# with open('lenthgy','rb') as f:
#     doc_length = pickle.load(f)
#     print("长度列表载入成功")
```

Listing 4: 构建带有 tf 词频的倒排索引字典

这里面比较关键的一点就是用 pickle 包来保存和载入字典

```
def cosine_scores(inverted_index,doc_length,querys,k=3):
    scores = defaultdict(lambda :0.0)#保存分数
    query_terms=Counter(term for term in jieba.cut(querys)if len(term.strip
    ())>1)##将用户输入的自然语言进行分词
    for q in query_terms:
        w_tq = 1+np.log(query_terms[q])
        ##实现idf权重分数的步骤
        df1 = len(query0(inverted_index,collections,q))
        w_qidf = np.log(N/df1)
        w_tq = w_tq*w_qidf
        posting_list = get_posting_list(inverted_index,q)
        for posting in posting_list:
            w_td = 1+np.log(posting.tf)
            w_didf=np.log(N/df1)
            w_td = w_td*w_didf
            scores[posting.docid]+=w_td * w_tq
    results = [(docid,score/doc_length[docid])for docid,score in scores.
    items()]
    results.sort(key=lambda x:-x[1])
    return results


def retrieval_by_cosine_sim(inverted_index,collections,query,k=10000):
    top_scores = cosine_scores(inverted_index,doc_length,query,k=k)
    results = [(collections[docid],score)for docid,score in top_scores]
    return results
```

Listing 5: 基于余弦相似度计算倒排索引返回关键列表

```python
def evaluate(query:str) ->list:##改进改成元素是列表的词典
    doclist = retrieval_by_cosine_sim(inverted_index,collections,query)
    docnum=[]
    urls = []
    titles=[]
    for doc,num in doclist:
        doc = int(doc[0:-4])
        docnum.append(doc)
    for doc in docnum:
        index1 = webnum.index(doc)
        flag1 = webs[index1].find(':')
        url = webs[index1][flag1 + 1:-1]
        urls.append(url)
        try:
            index2 = namenum.index(doc)
            flag2 = names[index2].find(':')

            title = names[index2][flag2 + 1:-1]
            titles.append(title)
        except:
            titles.append(" ")
    urls2 = list(set(urls))
    urls2.sort(key = urls.index)##这个是最终要的列表
    titles2 = list(set(titles))
    titles2.sort(key = titles.index)
    results = []
    for i in range(20):
        try:
            results.append({'title':titles2[i],'url':urls2[i]})
        except:
            return results
    return  results
```

Listing 6: 实现 WebUI 接口的 evaluate 函数

这一步要注意的是，返回的结果是一个以字典为元素的列表，而字典的值为 title 和 url

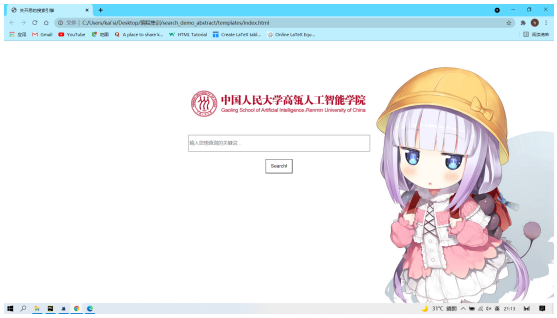

图 1: 搜索引擎搜索界面

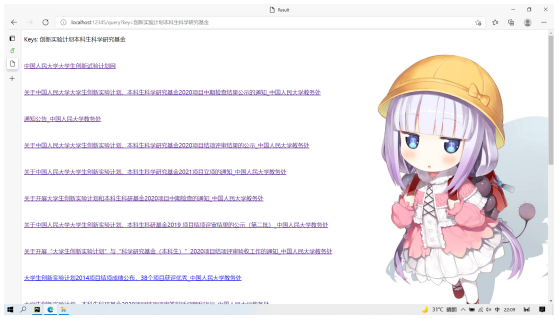

图 2: 搜索引擎搜索结果

4 界面展示

5 实验感想

这次小型搜索引擎的制作是我在大学阶段制作的第一个项目。整整六天时间完成了这个小项目对我来说十分有意义。首先我要感谢毛佳昕老师以及助教师兄们，能力有限的我们遇见了各种各样的问题，老师以及助教师兄总是能够不厌其烦地为我们解答，没有老师以及师兄们的帮助我们可能很难顺利完成这项任务。

这个小小的搜索引擎可能在以后看来只是一个非常简单的项目，但它带给我的提升也远远不是其他课程所能相比的，对于整体思路的把控，常见算法在实际中的应用，还有遇见莫名奇妙的 bug 时我们应该怎么应对，种种能力就是应该在不断的实践中去获取，这次项目也是我在未来科研或者做其他项目的一个初体验。作为一名准高瓴人，也是高瓴人工智能学院的第

一批本科生，我今后更应该多多实践，实践中提高自己的能力，希望以后的自己面对再大的任务也能明白每一步该做什么，面对再多 bug 能够心平气和。