小型搜索引擎的制作

DESIGNED BY 关开思

中国人民大学高瓴人工智能学院
Gaoling School of Artificial Intelligence，Renmin University of China

CONTENTS

/01

效果展示

关开思的搜索引擎

文件 | C:/Users/kai'si/Desktop/编程集训/search_demo_abstract/templates/index.html

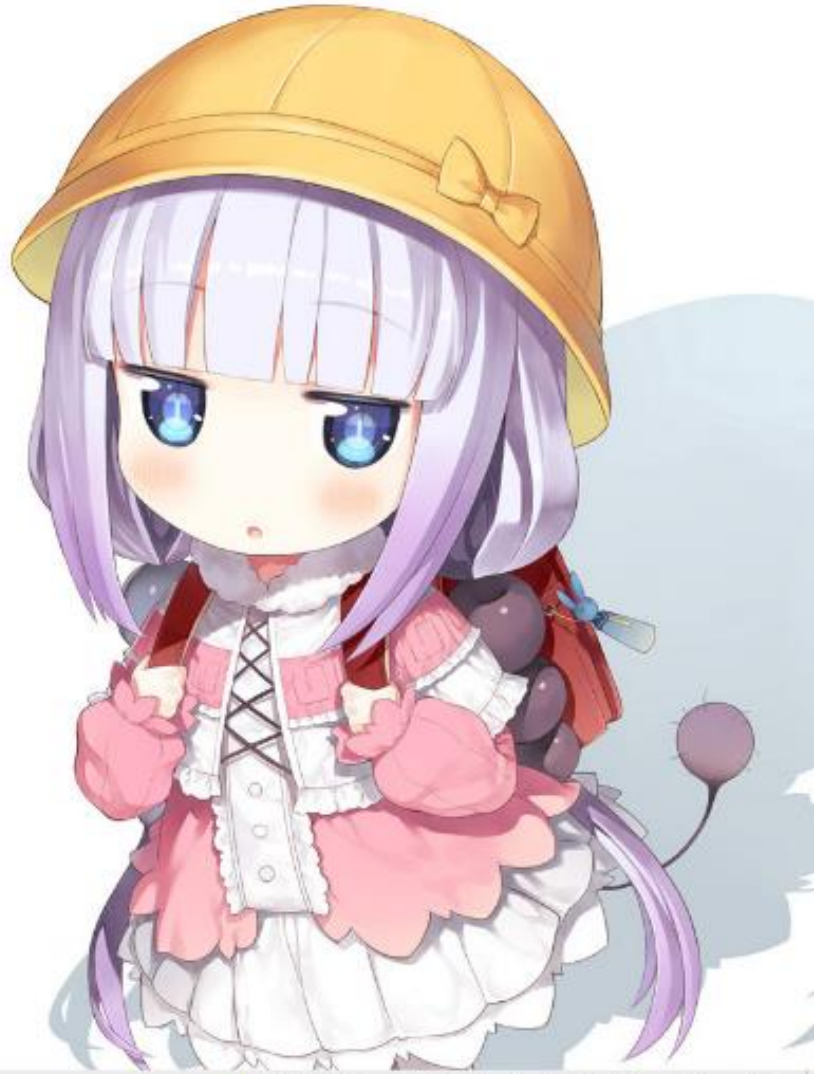
应用 Gmail YouTube 地图 A place to share k... HTML Tutorial Create LaTeX tabl... Online LaTeX Equ... 阅读清单

中国人民大学高瓴人工智能学院

Gaoling School of Artificial Intelligence，Renmin University of China

输入您想查询的关键词

Search!



31°C 晴朗 英 21:13

Keys: 创新实验计划本科生科学研究基金

中国人民大学大学生创新试验计划网

关于中国人民大学大学生创新实验计划、本科生科学研究基金2020项目中期检查结果公示的通知_中国人民大学教务处

通知公告_中国人民大学教务处

关于中国人民大学大学生创新实验计划、本科生科学研究基金2020项目结项评审结果的公示_中国人民大学教务处

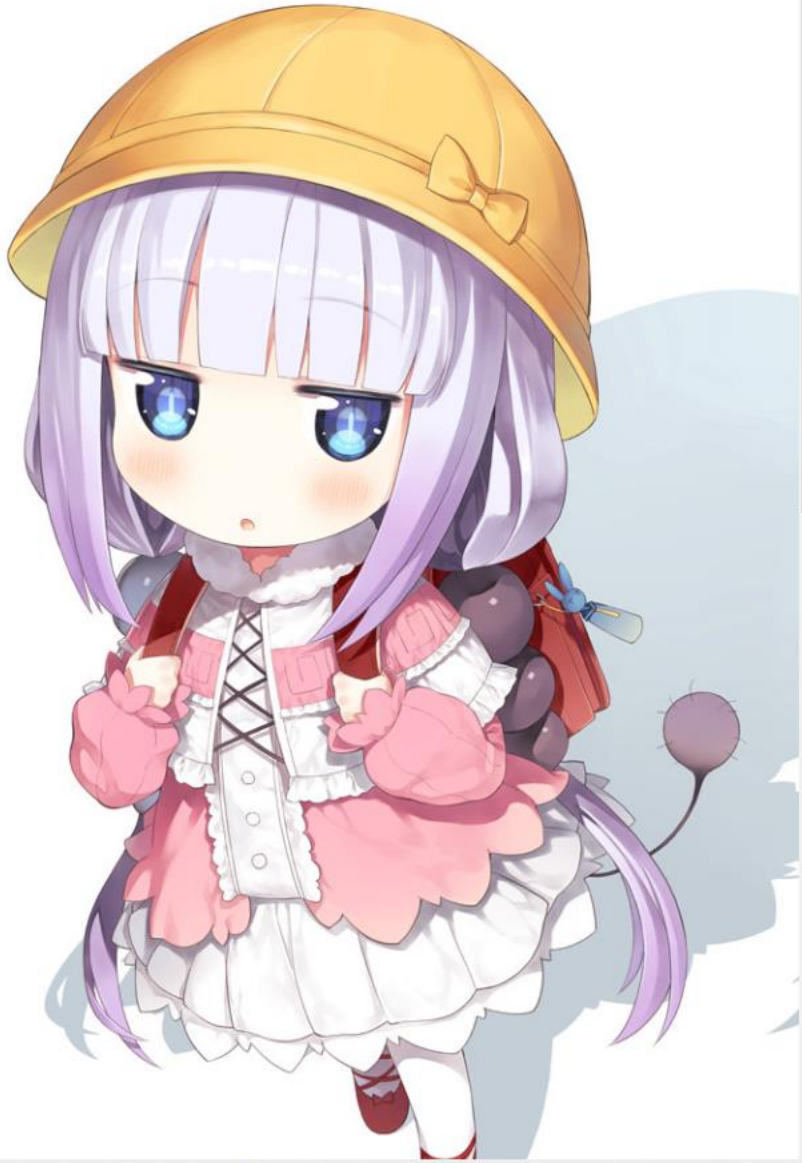
关于中国人民大学大学生创新实验计划、本科生科学研究基金2021项目立项的通知_中国人民大学教务处

关于开展大学生创新实验计划和本科生科研基金2020项目中期检查的通知_中国人民大学教务处

关于中国人民大学大学生创新实验计划、本科生科研基金2019 项目结项评审结果的公示（第二批）_中国人民大学教务处

关于开展“大学生创新实验计划”与“科学研究基金（本科生）”2020项目结项评审验收工作的通知_中国人民大学教务处

大学生创新实验计划2014项目结项成绩公布，38个项目获评优秀_中国人民大学教务处



/02

整体思路

整体思路

1

爬虫获取网页

2

构建倒排索引

3

基于余弦相似度排序

4

Web UI的制作

爬虫获取网页

爬虫主要采用requests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Beautiful-Soup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

构建倒排索引

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

基于余弦相似度排序

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：$W_{t,d}=(1+\log tf_{t,d})*\log(N/df_t)$,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：$Score=\sum_{t\in q\cap d} tf.idf_{t,d}$,**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。**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

Web UI 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。

/03

关键步骤代码

爬虫终止后继续 （pickle包）

```
queue = []
with open('queue','rb') as f:
    queue = pickle.load(f)
    print("队列载入成功")
used_urlset = set()
for url in input_urls:
    queue.append(url)
```

```
def savedic(dic):
    with open('url.txt','w',encoding='utf-8') as f:
        for key,value in dic.items():
            f.write(str(key))
            f.write(':')
            f.write(value)
            f.write('\n')
```

```
##保存URL文件
with open('queue', 'wb') as f:
    pickle.dump(queue, f)
    print('队列保存成功')
```

爬虫部分
BFS

```
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)
    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) + '.htm
        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
##全部代码完成
```

布尔值查询中and和or的实现

```python
def intersection(l1,l2):
    answer = []
    p1,p2 = iter(l1),iter(l2)
    try:
        docID1,docID2 = next(p1),next(p2)
        while True:
            if docID1 == docID2:
                answer.append(docID1)
                docID1,docID2 = next(p1),next(p2)
            elif docID1<docID2:
                docID1 = next(p1)
            else:
                docID1 = next(p2)
    except StopIteration:
        pass
    return answer
##实现or逻辑的归并
def merge(l1,l2):
    l,r = 0,0
    answer = []
    while l<len(l1) and r < len(l2):
        if l1[l]<l2[r]:
            answer.append(l1[l])
            l+=1
        else:
            answer.append(l2[r])
            r+=1
    answer+=l1[l:]
    answer+=l2[r:]
    return answer
```

特殊元素字典的保存

Pickle可以用来保存特殊的字典列表之类，并且可以不用转换成其它形式就可以自由的保存和载入
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

```
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)
...
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')

print("start loading:cost time about 10 seconds")
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)
```

/04

扩展功能的实现

tf-idf加权的分计算

N为文件的总数
利用标准化之后向量的
内积就是余弦值

```
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)#
    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))
        if df1!=0:
            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
        else:
            pass

    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
```

网页的优化

更改了表单的大小以及位置，设置背景图片，添加图片，并调节图片参数，注意这里面的图片一定要写相对路径，因为web server在python语言下无法解析绝对路径

```
    <meta charset="UTF-8">
    <title>关开思的搜索引擎</title>
    <style>
body {
  background-image: url("/static/456.jpg");
  background-repeat: no-repeat;
  background-attachment: fixed;
  background-size: cover;
}
</style>
  </head>
  <body>
  <br>
  <br><br><br><br><br>
    <div align="center"> <img src="/static/GSAI.png" style="..."></div>
  <br>
  <br>
    <div>
      <form action="query" method="get">
        <div style="...">
          <input style="..." type="text" name="key" placeholder="输入您想查询的关键词...">
        </div>
        <br>
        <div style="...">
          <button style="..." type="submit">Search</button>
        </div>
      </form>
    </div>
  </body>
</html>
```

/05

总结与反思

总结

- 比较顺利的完成了任务，利用pickle库等方式保存倒排索引词典，能高很好的提高了我的速度
- 在爬虫方面存在很多相同的页面，在爬虫实现上需要有所改进，网页之间的重要性也是不同的
- 在分数计算上，关于全权重的算法我认为需要更加改进，比如说出现在文章的不同位置权重也应该不同
- 网页设计上，事件原因设计比较简陋，可以用CSS优化界面

改进思考，如何判断网页重要性

- 原代码，只考虑了与种子文件的关系，和在种子url中出现的顺序
- 思考改进：搜索了以下资料，要根据与种子url和网页链接其他网页数量排序

思考改进

建立一个矩阵$\boldsymbol{H}$， 称为超链矩阵(hyperlink matrix)

$$\boldsymbol{H}(i,j)=\begin{cases}\dfrac{1}{l_j}, 若P_j \in \mathbb{B}_i \\ 0, 上述条件不成立\end{cases}$$

- 所有的元都是非负的
- 除非对应这一列的网页没有任何链接， 它的每一列的和为 1。

定义网页重要性向量$\boldsymbol{I}=[I(P_j)]$

$$I(P_i)=\sum_{j\in\mathbb{B}_i}\frac{I(P_j)}{l_j} \quad \longrightarrow \quad \boldsymbol{I}=\boldsymbol{HI}$$

THANKS

Thanks for your listening