教师搜索引擎

史恩扬 1711373

2019/12/4

摘要

通过爬虫爬取南开大学所有教师信息页，对其建立索引，并进行 PageRank 分析，通过 Web 页面进行检索。

关键字： Spider PageRank Search Engine

1 工程简介

本工程可划分为三个部分（见图1），分别为网络爬虫，索引构建及链接分析，Web 搭建。

最终是要实现一个可以对南开教师进行检索的搜索引擎，搜索结果包括教师对应的个人介绍页以及出现的其他网页。并且通过锚文本分析，以及 $BM25$ 算法对匹配的结果进行排序。

因为爬取的网站链接结构的原因，此处的 PageRank 的排序对于用户搜索结果并没有较好的优化效果，因此只将其实现并进行可视化展示。

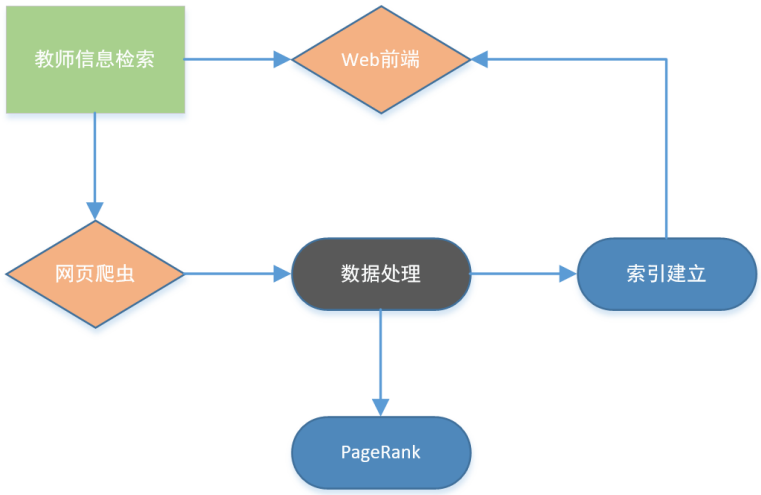

图 1: 工程结构

2 网页爬虫

通过 Python 支持的 Scrapy 框架编写网页爬虫。我们需要将南开内网中特定的域名下的所有网页进行无差别的连续爬取，因此选用 Scrapy 的 Crawl 模板。

为了提高爬取的速率和准确率，将各个学院的域名加入 $allow_set$ 中，将 bbs，软件之家等资源网站加入 $deny_set$ 中。制定爬取规则。

对每个符合规则的网页提取该网页的 title，url，纯文本，链接及链接附近的文本，将这些信息存入 item 中，并将 item 传给 pipeline。

```
class InfoSpiderSpider(CrawlSpider):
name = 'info_spider'
allowed_domains = ["nankai.edu.cn"]
start_urls = ['https://www.nankai.edu.cn/212/list.htm']
allow_set = ("wxy\.nankai\.edu\.cn",
             "history\.nankai\.edu\.cn",
             "phil\.nankai\.edu\.cn",
             "law\.nankai\.edu\.cn",
             "zfxy\.nankai\.edu\.cn",
             "fcollege\.nankai\.edu\.cn",
             "cm\.nankai\.edu\.cn",
             "economics\.nankai\.edu\.cn",
             "bs\.nankai\.edu\.cn",
             "sms\.nankai\.edu\.cn",
             "physics\.nankai\.edu\.cn",
             "chem\.nankai\.edu\.cn",
             "sky\.nankai\.edu\.cn",
             "cc\.nankai\.edu\.cn",
             "ceo\.nankai\.edu\.cn",
             "env\.nankai\.edu\.cn",
             "medical\.nankai\.edu\.cn",
             "cs\.nankai\.edu\.cn",
             "hyxy\.nankai\.edu\.cn",
             "tas\.nankai\.edu\.cn",
             "teda\.nankai\.edu\.cn",
             "tedabio\.nankai\.edu\.cn",
             "iap\.nankai\.edu\.cn",
             "esd\.nankai\.edu\.cn",
             "pharmacy\.nankai\.edu\.cn",
             "finance\.nankai\.edu\.cn",
             "mse\.nankai\.edu\.cn",
             "stat\.nankai\.edu\.cn",
             "www\.riyan\.nankai\.edu\.cn",
             "ifd\.nankai\.edu\.cn",
             "cyber\.nankai\.edu\.cn",
             "ai\.nankai\.edu\.cn")
deny_set = ('bbs\.nankai', 'career\.nankai', 'international\.nankai', '
news\.nankai', 'xxb\.nankai', 'std\.nankai',
            'soft\.nankai', 'weekly\.nankai', 'movie\.nankai', 'bt\.
nankai', 'www\.nankai', 'chinaeconomy\.nankai',
            'paper\.lib\.nankai', 'soft\.nankai',
            'jwc\.nankai', 'bt\.nankai', 'jw\.nankai', 'movie\.nankai',
'opac\.lib\.nankai', 'xxgk\.',
            'nkuefnew\.nankai','lbs\.nankai','ibs\.nankai','lib\.nankai
')
rules = (
```

```
43          Rule(LinkExtractor(allow=allow_set, deny=deny_set, unique=True),
    callback='parse_item',
44              follow=True),
45          Rule(LinkExtractor(allow=allow_set, unique=True, deny=deny_set),
    follow=True),
46      )
47
48      def parse_item(self, response):
49          item = SpiderItem()
50          text_XPath = "//text()"
51          href_XPath = "//@href"
52          title_XPath = "/html/head/title/text()"
53          anchor = []
54          x = LinkExtractor(allow=self.allow_set, deny=self.deny_set, unique=
    True, attrs=['href', 'text()'])
55          links = x.extract_links(response)
56          for link in links:
57              anchor.append((link.url, link.text))
58          item["href"] = response.xpath(href_XPath).extract()
59          item["my_url"] = response.request.url
60          item["text"] = response.xpath(text_XPath).extract()
61          item["title"] = response.xpath(title_XPath).extract()
62          item["anchor"] = anchor
63          return item
```

为了提高爬取速度，对于各个网页信息的过滤，规范化操作在 pipeline 中并行进行。将纯文本中的特殊符号剔除，仅保留文字信息，并对其中是否包含电子邮件地址，" 教授"，" 讲师"，" 职称" 等信息，以及网络层次（教师信息网页不会在第一层）作为过滤规则，将符合的网页信息以及对应的 ID 存在 json 文件中。为了后续的数据处理，将 id 与 url 对应的哈希表也存在 json 文件中。

```
64      class SpiderPipeline(object):
65      id = 0
66      url_dic = {}
67
68      def __del__(self):
69          json_str = json.dumps(self.url_dic, ensure_ascii=False)
70          path = "data/url2id.json"
71          with open(path, "w", encoding="utf-8") as json_file:
72              json_file.write(json_str)
73
74      def process_item(self, item, spider):
75          href = []
76          for i in item["href"]:
```

```
77              if (i == '#' or i.find('javascript') != -1 or i.find('.css') !=
    -1):
78                  continue
79              if (i.find("http://") == -1):
80                  i = parse.urljoin(item["my_url"], i)
81                  href.append(i)
82              else:
83                  href.append(i)
84          text = ""
85          for i in item["text"]:
86              x = i.strip()
87              if (x != "" and x.find("    ") == -1 and x.find("javascript") ==
    -1):
88                  text += x + ' '
89          pattern = r'[a-zA-Z0-9_-]+@[a-zA-Z0-9_-]+(\.[a-zA-Z0-9_-]+)'
90          match = re.search(pattern, text, re.M | re.I)
91          if parse.urlparse(item["my_url"]).path.count("/") > 1 and match !=
    None and text.find("招聘简章") == -1 and text.find("公示") ==-1 and
    text.find("讲座") ==-1 and text.find("讲堂") ==-1 :
92              if text.find("简介") != -1 or text.find("职称") != -1 or text.
    find("教授") != -1 or text.find("研究员") != -1 or text.find("个人") !=
     -1 or text.find("姓名") != -1:
93                  dic = {}
94                  dic["id"] = self.id
95                  dic["my_url"] = item["my_url"]
96                  dic["text"] = text
97                  dic["title"] = item["title"]
98                  dic["href"] = href
99                  dic["anchor"] = item["anchor"]
100                 self.url_dic[item["my_url"]] = self.id
101                 self.id += 1
102                 json_str = json.dumps(dic, ensure_ascii=False)
103                 path = "data/" + str(dic["id"]) + ".json"
104                 with open(path, "w", encoding="utf-8") as json_file:
105                     json_file.write(json_str)
106         return item
```

3 数据处理

3.1 索引建立

借助 Python 提供的 Whoosh 模块结合 jieba 模块提供的分词工具快速构建索引。索引的 Schema 包含网页对应的 ID，url，text，title，anchor

text。其中 text 和 anchor text 是作为检索部分，url，title 会作为结果返回。

程序会遍历网页爬虫获取到的所有 json 文件，收集所有锚文本储存在一个字典中，再次遍历，通过自身 ID 获取本网页的锚文本以及 json 文件中的其余信息添加到索引中。

```
107     def index_init(rootdir):
108     analyzer = ChineseAnalyzer()
109     schema = Schema(id=ID(stored=True, unique=True), url=ID(stored=True),
        text=TEXT(analyzer=analyzer),
110                     anchor_text=TEXT(analyzer=analyzer), title=TEXT(stored=
        True))
111     with open('spider/data/url2id.json', encoding='utf-8') as file:
112         url_dic = json.load(file)
113     if not os.path.exists('searching_engine/index'): # 如果目录index不存在
        则创建
114         os.mkdir('searching_engine/index')
115     create_in("searching_engine/index", schema) # 按照schema模式建立索引目
        录
116     index = open_dir("searching_engine/index") # 打开该目录一遍存储索引文
        件
117     writer = index.writer()
118     anchor = {}
119     print("begin init anchor dic")
120     for root, dirs, files in os.walk(rootdir):
121         for name in files:
122             # 收集所有anchor
123             fp = os.path.join(root, name)
124             fp = fp.replace('\\', '/')
125             with open(fp, encoding='utf-8') as file:
126                 dic = json.load(file)
127             if(dic["anchor"].__len__!=0):
128                 for a in dic["anchor"]:
129                     id = url_dic.get(a[0])
130                     if(id):
131                         t = anchor.get(id)
132                         if(t):
133                             t += a[1]
134                         else:
135                             anchor[id] = a[1]
136     print("begin init index")
137     counter = 0
138     for root, dirs, files in os.walk(rootdir):
139         for name in files:
140             # 遍历所有文件
141             fp = os.path.join(root, name)
142             fp = fp.replace('\\', '/')
```

```
143             with open(fp, encoding='utf-8') as file:
144                 dic = json.load(file)
145             my_anchor = anchor.get(chr(dic["id"]))
146             # 构建index
147             if(my_anchor):
148                 writer.add_document(id=chr(dic["id"]), url=dic["my_url"],
149                                     text=dic["text"], title=dic["title"],
    anchor_text=my_anchor)
150             else:
151                 writer.add_document(id=chr(dic["id"]), url=dic["my_url"],
152                                     text=dic["text"], title=dic["title"])
153             counter += 1
154             if(counter==1000):
155                 counter = 0
156                 print(fp)
157     writer.commit()  # index构建完成
```

3.2 PageRank

初始情况下，对于 PageRank 的实现，只需要通过矩阵的乘法，在设置一个阈值的基础上进行迭代，其实质过程是马尔科夫链模型。公式如下

$$r_j^{(t+1)} = \sum_{i \to j} \left(\frac{r_i^{(t)}}{d_i} \right)$$

然而，这对于一些特殊情况无法得到一个理想的结果。

3.2.1 稀疏矩阵

显而易见网页之间的连接矩阵是一个稀疏矩阵，因此以矩阵方式存储会大量浪费存储空间，因此仅保存每条边的信息。

```
158     threshold = 1.0e-6
159     most_rank = 100
160     ...
161     dt = np.dtype([('from', int), ('to', int)])
162     A = np.empty((0), dtype=dt)
163     NUM = len(os.listdir(rootdir))
164     ...
165     for root, dirs, files in os.walk(rootdir):
166         for name in files:
167             #  遍历所有文件
168             fp = os.path.join(root, name)
169             fp = fp.replace('\\', '/')
```

```
            with open(fp, encoding='utf-8') as file:
                dic = json.load(file)
            A = np.append(A, np.array([[dic["id"], to_url)]], dtype=dt))
    ...
    v_new = np.ones(NUM) * 1 / NUM
    v_old = v_new
    B = v_new
    rank = 1
    while ((np.sum(np.abs(v_new - v_old)) > threshold or rank == 1) and rank
     < most_rank):
    v_old = v_new
    x_old = A[0]
    start = end = 0
    v_new = np.zeros(NUM)
    for x in A:
        # 对于从同一点出发的边统一处理，仅需从A中读取从某一点出发的所有边即
    可，实现了分块矩阵
        if (x[0] != x_old[0]):
            for i in range(start, end):
                v_new[A[i][1]] += v_old[x_old[0]] / (end - start)
            start = end
            x_old = x
            end += 1
        else:
            end += 1
    ...
```

3.2.2 Spider Trap

对于一些图会存在 Dead End 的情况，他们会吸收图中传递的值并在其中循环传递，不再向外传递。因此，我们可以加上一个 β 值对于 dead end 进行 random teleport。使得我们的计算方式变为

$$r_j^{(t+1)} = \sum_{i \to j} \left(\beta \frac{r_i^{(t)}}{d_i} \right) + (1+\beta)\frac{1}{N}$$

```
    beta = 0.85
    ...
    v_new = beta * v_new + (1 - beta) * B
```

3.2.3 Dead End

对于一些图会存在 Dead End 的情况，他们会吸收图中传递的值并且这些值将消失。为了解决 Dead End 问题，将出度为 0 的 Dead End 中的值加和并重新分配给各个点。

```
    # 将边的集合A以出发点的序号排序便于后续计算
    A = np.sort(A, order='from')
    # 利用w记录各点出度是否为零，便于发现dead end
    w = np.zeros((NUM))
    for a in A:
        w[a[0]] = 1
    ...
    index = 0
    sum = 0
    # 对于dead end进行random teleport
    for x in w:
        if (x == 0):
            sum += v_old[index]
            index += 1
        else:
            index += 1
    v_new += sum * np.ones(NUM) * 1 / NUM
```

3.2.4 PageRank 可视化

通过 Python 提供的 Pyechart 模板中的 Graph 类生成网页连接图，为了避免过多的节点导致过于混乱，此处只选取 PageRank 值 TOP40 的网站进行绘图。因为节点的 PageRank 的值较小且差异不大，此处对其进行如下操作

$$node_size = 1.2^{value*1000}$$

```
    def drawing_pagerank(rootdir):
    from pyecharts import options as opts
    from pyecharts.charts import Graph
    with open('spider/data/url2id.json', encoding='utf-8') as file:
        url_dic = json.load(file)
    nodes = []
    links = []
    top40 = {}
    t_value = np.load('searching_engine/PageRank/PageRank_value.npy')
    t = np.load('searching_engine/PageRank/PageRank.npy')[:40]
    for id in t:
```

```
225         with open('spider/data/doc/'+str(id)+'.json', encoding='utf-8') as
    file:
226             dic = json.load(file)
227         if (dic["title"].__len__() != 0):
228             top40[id] = dic["title"][0] + "     " + dic["my_url"]
229         else:
230             top40[id] = dic["my_url"]
231     for id in t:
232         with open('spider/data/doc/'+str(id)+'.json', encoding='utf-8') as
    file:
233             dic = json.load(file)
234         if(dic["title"].__len__()!=0):
235             nodes.append({"name": dic["title"][0]+"     "+dic["my_url"],
236                           "symbolSize": pow(1.2, t_value[id]*1000), "value":
     pow(1.2, t_value[id]*1000)})
237         else:
238             nodes.append({"name": dic["my_url"], "symbolSize": pow(1.2,
    t_value[id]*1000), "value": pow(1.2, t_value[id]*1000)})
239         for i in dic["href"]:
240             to_url = url_dic.get(i)
241             target = top40.get(to_url)
242             if (to_url and target!=None):
243                 links.append({"source": top40[dic["id"]], "target": target})
244     print("begin drawing")
245     graph = Graph().add("", nodes, links, repulsion=2000).set_global_opts(
    title_opts=opts.TitleOpts(title="PageRank for Top40"))
246     graph.render("searching_engine/PageRank/PageRank.html")
247     graph.render("web/templates/PageRank.html")
```

3.3 搜索端口

对于索引进行读取，并对传入的信息在 text，title 以及 anchor text 部分检索，并对结果按照 $BM25$ 算法进行排序。并且可以对搜索词进行纠正（见图2），用空格连接搜索词可进行与操作（见图3），" 域名:xxx" 可对特定域进行（见图4）。

```
248     def search_for(content):
249     index = open_dir("searching_engine/index")
250     with index.searcher() as searcher:
251         parser = MultifieldParser(['title', 'anchor_text', 'text'], index.
    schema, fieldboosts={''anchor_text'': 1.5})
252         query = parser.parse(content)
253         corrected = searcher.correct_query(query, content)
254         if corrected.query != query:
```

```
255             q = parser.parse(corrected.string)
256             print(corrected.string)
257         else:
258             q = query
259         result = searcher.search(q, limit=None)
260         res = []
261         for r in result:
262             x = dict(r)
263             # res.append([ord(x["id"]), x["url"]])
264             res.append([x["title"], x["url"]])
265         return res
```

图 2: 包含纠错机制的查询

图 3: 多个关键词进行布尔查询

3.3.1 $BM25$ 算法

$BM25$ 算法是一种常见用来做相关度打分的公式，思路比较简单，主要就是计算一个 query 里面所有词和文档的相关度，然后再把分数做累加

图 4: 按域查找

操作。公式如下：

$$
\begin{aligned}
Score(Q,d) &= \sum_{i}^{n} W_i \cdot R(q_i,d) \\
W_i &= IDF(q_i) = log\frac{N+0.5}{n(q_i)+0.5} \\
R(q_i,d) &= \frac{f_i \cdot (k_1+1)}{f_i+K} \cdot \frac{qf_i \cdot (k_2+1)}{qf_i+k_2} \\
K &= k_1 \cdot (1-b+b \cdot \frac{dl}{avgdl})
\end{aligned}
\tag{1}
$$

其中 $R(q_i,d)$ 是查询语句 query 中每个词 q_i 和文档 d 的相关度值，W_i 是该词的权重，一般情况下为 IDF(InverseDocumentFrequency) 值，即逆向文档频率,N 是文档总数，$n(q_i)$ 是包含该词的文档数，0.5 是调教系数，避免 $n(q_i)=0$ 的情况。log 函数是为了让 IDF 的值受 N 和 $n(q_i)$ 的影响更加平滑。k_1,k_2,b 都是调节因子，一般 $k_1=1,k_2=1,b=0.75$。式中 qf_i 为词 q_i 在查询语句 query 中的出现频率，f_i 为 q_i 在文档 d 中的出现频率。由于绝大多数情况下一条简短的查询语句 query 中，词 q_i 只会出现一次，即 $qf_i=1$。dl 为文档 d 的长度，$avgdl$ 为所有文档的平均长度。

4 Web 界面

借助 Python 支持的 Djiango 框架，快速搭建 Web 页面。首页包括搜索引擎名称以及搜索框（见图5）。

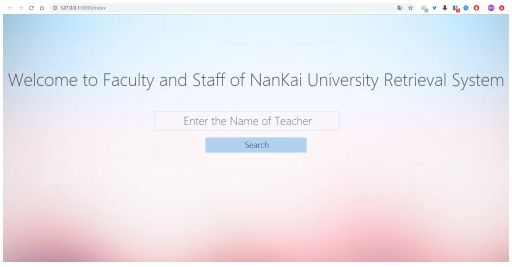

图 5: 首页

引入 search engine 中的 search for 函数，将 GET 获取到的搜索内容传入，并将结果列表和搜索内容作为参数传入 search.html。检索结果会以该网页的 title 形式显示，点击即可访问该网页（图6）。结果页面可通过 Back 按钮返回首页，或在搜索框中继续搜索。

图 6: 搜索” 王刚” 并打开搜索结果

通过访问 pagerank（见图7）可以看到 PageRank 排序 TOP40 的网络节点之间的链接结构，节点的大小表示了该网页的权重。

图 7: PageRank 可视化