《Python程序设计》大作业

实验报告

实验题目： 高校就业信息网爬取

姓名： 林亦旸

学号： 2020212142

日期： 2022/1/07

1. **需求分析**

本次作业分为两个部分进行。第一部分，是爬取北邮就业信息网，电子科技大学研究生就业信息网和西电就业信息网的招聘信息。第二部分是对爬取下来的数据做数据处理，得到最受关注几条的招聘信息。

首先，对需要爬取的网站进行页面分析。北邮就业信息网是动态页面，需要用selenium模拟用户打开页面或用json文件爬取。发现北邮就业信息网的翻页是通过脚本控制的，不能靠修改url实现翻页。而我们需要进入招聘信息二级页面进行爬取，这样会导致翻页无法正常进行。除非用一个浏览器爬取目录，另外一个爬取二级页面。但这样做多占用系统资源。故在源代码中查找读取json文件的脚本，找到获取json的方法。用json文件爬取，不仅速度快，内容也好解析，对网站服务器造成的负载也小。因此对于成电就业网我也采取了这种爬取方式。西电就业网因为在之前作业中爬取过，直接对之前的代码做修改即可完成任务。

之后，需要对爬取的数据进行整理分析。Pandas可以高效的完成这个任务。整理数据的第一步要依据帖子主题去重，去重的时候要注意累计阅读量。第二步要合并三个表格，此时先暂时去掉北邮的岗位数量。这样就得到了合并过的数据。

之后进行分析处理。需要先标注雇主类型，并进行匹配。由于只有帖子名称，用机器学习的方法进行分类会出错，考虑直接在合并出来的Excel表格里搜索关键字进行人工标注。用pandas对dataframe进行排序，得到需要的TOP榜单。最后在报告中对数据进行总结。

1. **模型设计**

由于时间仓促，没有对所有数据做出雇主类型分类。在分析招聘数据TOP榜单时，对雇主进行行业分析。

1. **系统设计**

分为两个脚本进行编写，一个是爬虫脚本，一个是数据处理模块脚本

1. 对爬虫的设计

采用Scrapy框架，结合selenium，json模块进行爬取。

item.py中设置四个item：

*class* Jobtop10Item(*scrapy*.*Item*):

    job\_title = scrapy.Field()  # 招聘帖子的主题

    job\_date  = scrapy.Field()  # 招聘帖子的发帖时间

    job\_views = scrapy.Field()  # 招聘帖子的浏览次数

    job\_nums  = scrapy.Field()  # 招聘帖子的职位数量

pipeline.py中写好了对item的处理，将其写入对应csv

setting.py中设置了多爬虫启动和中间件

middlewares.py中，设计下载中间件。这边和之前作业的设计不一样。之前的作业在启动多爬虫时会多开几个没有执行任务的浏览器。这是因为scrapy对每个spider都实例化了一个下载中间件类。按照之前的作业代码，n个爬虫会开启n\*n个浏览器，浪费系统资源。并且之前作业中爬虫get\_driver调用的是类方法，这样做最终调用的都是最后一个被实例化的下载中间件类。可不可以在传给spider的response里面找到scrapy为这个spider实例化的中间件呢？可以，但是需要运行时观察变量，找到自定义的中间件在中间件列表里的下标。这样写代码不利于维护，因为一旦改变中间件顺序，就得改spider。因此，我认为应该在spider初始化的时候进行driver的声明，变成spider的类对象，下载中间件使用spider.driver.get()方法进行页面打开，并返回response。具体实现如下：

spider.py中的实现

@*classmethod*

*def* from\_crawler(*cls*, *crawler*, \**args*, \*\**kwargs*):

# 在crawler初始化的时候被调用一次

        spider = *super*(XDUJOBSpider, *cls*).from\_crawler(

            crawler, \*args, \*\*kwargs)

        # 创建浏览器

        spider.driver = webdriver.Firefox()

        crawler.signals.connect(spider.spider\_closed,

*signal*=signals.spider\_closed)

        return spider

*def* spider\_closed(*self*, *spider*):

        spider.driver.close()  # 关闭浏览器

        print("==========爬虫结束！==========")

        spider.logger.info('Spider closed:%s', spider.name)

middlewares.py中的实现：（只需要实现process\_request函数）

*def* process\_request(*self*, *request*, *spider*):

        re\_method = request.meta.get('method', 'default')

        if re\_method=='default': #selenium爬取

            spider.driver.get(request.url)

            time.sleep(3)  #等待页面加载完成

            return scrapy.http.HtmlResponse(*url*=request.url,

*body*=spider.driver.page\_source.encode('utf-8'),

*encoding*='utf-8', *request*=request, *status*=200)

        elif re\_method == 'stastic':

            return None #静态爬取，交给其他中间件处理

这边为了配合静态爬取json文件，在爬虫发出request的时候会在meta里传一个参数method，代表静态爬取还是selenium爬取

爬虫的实现（代码中包括了注释，不赘述功能）：

XDUJOB\_spider.py

*def* parse(*self*, *response*): #解析爬取的内容

        item = Jobtop10Item()

        #下一页的url

        next\_page\_href=response.css('li[class="next"]>a::attr(href)').extract()

        # 末尾页的url

        last\_page\_href=response.css('li[class="last"]>a::attr(href)').extract()

#确认现在页面不是最后一页

#在尾页页的时候页面发生改变，尾页的链接在li[class="last "]下

        #按上面代码无法获取，由于只有一页，直接用len判断

        if response.url!=last\_page\_href and len(last\_page\_href) :

*self*.xidian\_next\_page = 'https://job.xidian.edu.cn' + next\_page\_href[0]

        else:

*self*.xidian\_next\_page = ''

        c\_page\_url\_list = response.css('ul[class="infoList"]>li:nth-child(1)>a')

        for job in c\_page\_url\_list:  # 循环打开和解析每个详情页

            driver=*self*.driver #用爬虫定义的driver

            driver.get('https://job.xidian.edu.cn' + job.css('a::attr(href)').extract()[0])

            time.sleep(2)#等待2秒钟

            item['job\_title'] = [driver.find\_element(

'css selector', 'div[class="title-message"]>h5').text][0]

            date\_text = driver.find\_element('css selector', 'div[class="share"]>ul>li:nth-child(1)').text

            date\_text=date\_text[date\_text.find('：') + 1:]

            if datetime.strptime(date\_text,'%Y-%m-%d %H:%M')

<datetime.strptime('2021-09-01 00:00','%Y-%m-%d %H:%M'):

*self*.xidian\_next\_page=''

 #到了设定时间，停止爬取。刚好可以利用下一页的url作为结束标志

                break

            item['job\_date'] = [date\_text][0]

            views\_text = driver.find\_element('css selector', 'div[class="share"]>ul>li:nth-child(2)').text

            item['job\_views'] = [views\_text[views\_text.find('：') + 1:]][0]

            item['job\_nums']=1

            yield item

        #处理完列表页的所有二级页面后，继续打开下一页进行抓取

        if *self*.xidian\_next\_page!='':

        #如果没有下一页了，停止爬取

            yield scrapy.Request(*self*.xidian\_next\_page, *callback*=*self*.parse)

BUPTJOB \_spider.py

*class* BUPTJOBSpider(*scrapy*.*spiders*.*Spider*):

    name="BUPTJOB"

    allowed\_domains = ["job.bupt.edu.cn"]

    # start\_urls = ['https://job.bupt.edu.cn/f/recruitmentinfo/ajax\_frontRecruitinfo?pageNo=']

    pagenum=1

*def* start\_requests(*self*):

        #由于start\_urls不能传静态爬取参数，用start\_requests

        yield scrapy.Request('https://job.bupt.edu.cn/f/recruitmentinfo/ajax\_frontRecruitinfo?pageNo=',

*callback*=*self*.parse, *meta*={'method': "stastic"})

*def* parse(*self*, *response*):

        data = json.loads(response.body)

        # 获取网页的json数据

        jobpost\_list = data['object']['list']

        print("[BUPT]now pagenum:"+*str*(data['object']['pageNo']))

        print("[BUPT]job len:"+*str*(len(jobpost\_list)))

        for jobpost in jobpost\_list:

            if datetime.strptime(jobpost["startTime"], '%Y-%m-%d %H:%M:%S') < datetime.strptime('2021-09-01', '%Y-%m-%d'):

                break

            id = jobpost['id']

            #获取下一页url，到二级页面爬取

            yield scrapy.Request(*url*=*f*'https://job.bupt.edu.cn/f/recruitmentinfo/ajax\_show?recruitmentId={id}',

*callback*=*self*.parse\_details, *meta*={'method': "stastic"})

#回调函数改变

*self*.n=data['object']['pageNo']+1

        if not data['object']['lastPage']:

            #判断尾页

            yield scrapy.Request(*url*=*f*'https://job.bupt.edu.cn/f/recruitmentinfo/ajax\_frontRecruitinfo?pageNo={*self*.n}',

*callback*=*self*.parse, *meta*={'method': "stastic"})

*def* parse\_details(*self*, *response*):

        #爬详情页，代码非常简单

        item = Jobtop10Item()

        data = json.loads(response.body)

        job = data['object']['recruitmentinfo']

        item['job\_title'] = job['title']

        item['job\_date'] = job['startTime']

        item['job\_views'] = job['browseNumber']

        item['job\_nums'] = job['positionNum']

        yield item

UESTCJOB\_spider.py

*def* parse(*self*, *response*):  # 解析爬取的内容，连二级页面都不用爬了

        data = json.loads(response.body)   # 获取网页的json数据

        jobpost\_list = data['list']

        print("[UESTC]now pagenum:"+*str*(data['page']))

        print("[UESTC]job len:"+*str*(len(jobpost\_list)))

        for jobpost in jobpost\_list:

            jobtime=jobpost["date"]

            jobtime = jobtime.replace("年", '-').replace("月", '-').replace("日", '')

            # print(jobtime)

            if datetime.strptime(jobtime, '%Y-%m-%d') < datetime.strptime('2021-09-01', '%Y-%m-%d'):

                return None

            item = Jobtop10Item()

            item['job\_title'] = jobpost['title']

            item['job\_date'] = jobtime

            item['job\_views'] = jobpost['viewcount']

            item['job\_nums'] = 1

            yield item

*self*.n = data['page']+1

        if *self*.n <= data['totalPage']:

            yield scrapy.Request(*url*='https://yjsjob.uestc.edu.cn/coread/listeminfo.action?page={}'.format(*self*.n),*callback*=*self*.parse, *meta*={'method': "stastic"})

最后运行出来的结果，北邮和成电在很短的时间内完成爬取（约五分钟），西电爬取了2个小时。如果动态页面能用json爬取，无疑是比较优秀的方案。

1. 对数据处理模块的设计

在文件夹中的dataprocess.ipynb 笔记本中，主要使用pandas模块。

首先进行数据预处理，爬取下来的标题可能有多余字符，删除。由于爬虫阶段对日期、浏览次数、职位数量等数据格式有进行过处理，不需要再处理。之后进行排序和去重。按照帖子名称进行排序，并且去重时累加阅读量。（注：因为之后的数据分析都不会再按照日期排序，所以合并后的数据也没有必要按日期顺序输出）

之后需要进行数据合并，使用pd.concat()方法。再次进行去重，此次去重还需要累加雇主发帖的学校数量。之后用将pd的ExcelWriter设置为openpyxl引擎，并对workbook进行操作，输出三校招聘信息和合并后的数据到JOBmerged.xlsx。

由于时间不足，没有标注数据，直接来到数据分析处理。用pandas对dataframe按照浏览量进行排序，得到三校的关注top20。筛选三校都发布招聘的雇主，由于数量并不多，排序后输出。将这些数据输出到JOBanalysis.xlsx中。

1. **数据分析**

最受北邮学生关注的招聘TOP20****

可以看到ICT相关包揽了前七。研究院和高校在TOP20的比例最高（从帖子主题并不能了解高校是要招聘博士后还是讲师还是研究员），说明北邮人除了大厂内推，还很喜欢去研究院和高校就业。有意思的是，并没有看到国企。

北邮招聘职位数量TOP10

可以看到ICT行业包揽全榜。说明ICT行业非常青睐北邮人，愿意给北邮人offer。



最受西电学生关注的招聘TOP20

****

最受成电学生关注的招聘TOP20

从第一名可以看出，成电学生就业时很关注地域。

****

在三校均发布招聘信息的雇主

可以看到几乎都是研究院和院校。可能的原因应该是应届生几乎通过内推渠道进入大厂，并不需要在就业信息网上进行查看消息。

****

1. **作业小结**

本次作业耗时比较久的环节在爬虫上，因为北邮网站的特性，我一直想寻找高效、节省资源的爬取方法，花费了大量时间。很遗憾的是我最后也没有找到用selenium的高效爬取方案。好在我了解到了json的爬取方式，直接静态爬取，高效省时。

最后爬取的数据很完整，格式也很好。但是这样的时间开销，让我也没有时间给数据做标记。最后的数据处理只能做到这个程度，也是这次作业一个很大的遗憾。最后感谢老师的这次作业让我综合运用了本学期学习的知识，运用了很多强大的库。这让我不需要像c++一样写很多算法提高程序效率，因为许多库的开发者都已经把效率提升得很高了。我只需要根据需求直接调用接口，就可以完成任务。Python的便捷之处或许就在于此。希望以后能用这样强大的工具为自己的生活增添一些乐趣。