**实习总结**

李诚

2017.11.6

在京东方实习期间，我主要工作是网络爬虫的维护，以及后面做的导诊机器人的自然语言处理工作。报告中提到的所有代码均已提交至github。

**一、网络爬虫维护部分**

1. **外部监控模块的搭建**

目的是对爬虫的状态进行监控，以确定爬虫程序是否在工作。具体做法是读取当前各个网站抓取的网页数量；与上一次的网页数量做对比，确定网页数量增量；将当前网页数量以及网页增量打包发送至个人邮箱。

程序的目录为project/app/kg/exter\_crawl\_monitor/src，其中，externalCrawlerMonitor.py为外部监控程序，myTimerRun.sh脚本为每天固定时间运行externalCrawlerMonitor.py程序。

mailSendNew(mailAddress, filePath)函数：

功能：读取file文件的内容，将file内容发送到mailAddress的邮箱；

比如：mailSendNew('chen\_lee@126.com',logPath)

1. **爬虫程序的优化**

网络爬虫用到的几个重要的Python模块：urllib.request，用于网页请求；re、BeautifulSoup用于网页解析。

网络爬虫的过程为：step1.从队列中取url；step2.抓取网页；step3.检查网页是否需要保存；step4.解析出网页中的url，判断url是否需要存入队列。step4中需要对url进行去重，之前是将历史url存入levelDB中，随着数据库中的ulr数量的增加，数据查找和存储的时间变长，对于一个网页解析出多个url，那么查找的时间也会很长，所以必须换套策略。

1. **url去重方式**

当前url去重的方式有很多种，这里举出三种，如下表1所示。Bloom Filter基本原理是位数组和Hash函数联合使用，所以存储和查找的效率比较高，但是这种方式存在小概率误判的问题，对于网络爬虫，当用这种方式可能出现少抓几个网页的问题，如表1所示，误判的概率是0.0000889，当需要查找10万个url时，可能存在9个url判断错误，也就是可能少抓取了9个url，少抓取几个网页对整个爬虫任务来说，基本没啥影响，所以采取Bloom Filter作为去重方式。

**表1 各种url去重方式原理及特点**

|  |  |  |
| --- | --- | --- |
| **方式** | **原理** | **特点** |
| url直接存入Hash表 | HashSet中放置的就是url的字符串，任何一个新的url首先在HashSet中进行查找，如果HashSet中没有，就将新的url插入HashSet，并将url放入待抓取队列。 | 这个方案的好处是它的去重效果精确，不会漏过一个重复的url。它的缺点是，很容易造成 Out Of Memory。因为随着抓取网页的增加，HashSet会一直无限制的增长。另外，网络中的很多url其实是很长的，有大量的url长度达到上百个字符。 |
| 压缩url存入Hash表 | 原始的url长度较大，考虑将原始url进行压缩再存入hashSet，可以用现成的方法对url进行压缩，比如直接用MD5对url做编码。MD5的结果是128bit也就是16byte的长度。相比之前估计url平均长度已经缩小了很多。 | 可以存储的历史url比直接用hashSet存储的url多，但是随着url增多，还是可能出现Out of Memory的情况，而且有可能两个不同的url的MD5值是相同的。 |
| Bloom Filter去重方式 | Bloom Filter基本原理是位数组和Hash函数联合使用。具体而言，Bloom Filter是一个包含了m位的位数组，数组的每一位都初始化为0，然后定义k个不同的Hash函数，每个哈希函数都可将集合中的元素映射到位数组的某一位。 | 对于已经在集合中的元素，通过5中的查找方法，一定可以判定该元素在集合中。对于不在集合中的元素，可能会被误判在集合中。有文献指出，哈希函数个数k取10，位数组大小m设为字符串个数n的20倍时，false positive发生的概率是0.0000889，这个概率已经很小了。 |

Python2.7中已经有了比较官方的Bloom Filter使用和安装文档，具体见：<http://axiak.github.io/pybloomfiltermmap/> ，在Python3.5中安装会出错，可以参考这篇文章安装：<http://blog.csdn.net/qq_16583687/article/details/77746644> 。

1. **爬虫功能的丰富与程序说明**

增加url定向抓取的功能，即给定一个url队列，只抓取这些url，用在准确知道目标url 是哪些，且目标url数量不多的情况。

爬虫程序在github工程目录：project/lib/dm/spider/minisb.py。对象MiniSpider初始化函数说明：

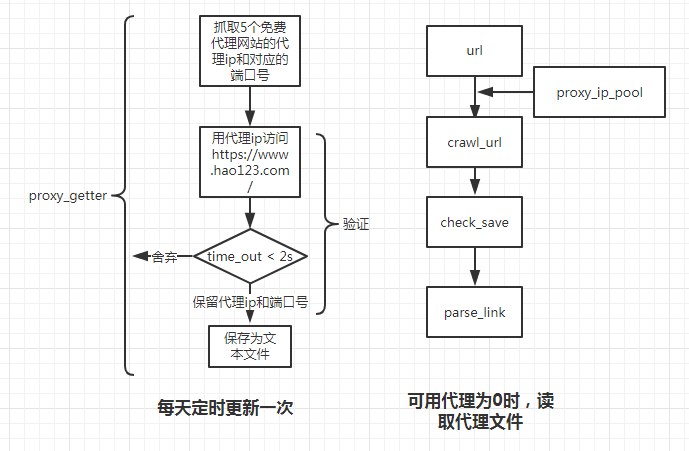
*def \_\_init\_\_(self, url\_db\_dir, result\_dir, entry\_list,proxy\_dir, white\_list = [], black\_list = [], download\_list = [], piror\_mode = 0, thread\_num = 1, time\_out = 1, time\_sleep = 1, queue\_max\_size = 1e6, url\_max\_size = 1e7, file\_max\_size = 1e10, debug = 0,run\_mode = 0,use\_proxy = False):*

对象初始化函数，proxy\_dir为代理IP的目录；run\_mode为0时是正常抓取模式，run\_mode为1时是定向抓取模式，需要提前在result\_dir目录下建立一个'%s.queue' % self.url\_db\_dir 文件，将目标url存入；use\_proxy为是否使用代理，如果为True则需要使用代理，False为不使用代理，若该网站没有封住本机ip，则没有必要开启代理IP。

1. **代理IP获取模块**

某些网站有反爬虫的策略，比如某些网站会统计每个用户请求的次数，当单个用户在某一时间段的请求次数超过一定值时，网站会拒绝该用户的之后的所有请求。当出现拒绝访问的情况时，有必要使用代理IP来隐藏自己的真实IP，当发出网页请求的时候，Request信号会先发送到代理服务器，由代理服务器来取回所需要的信息并传送回自己本地。

代理IP池模块的程序及思路参考：<https://github.com/jhao104/proxy_pool>，直接使用该程序会出现问题，所以自己改了一个简单的代理IP获取模块，基本思想如下图1所示。首先从5个免费代理网站抓取最新的代理ip及对应的端口号；由于这些代理IP很不稳定，需要对抓取的IP进行验证，即用代理IP访问同一个网站，如果访问成功且请求时间不超过2s则认为该代理IP可用；将结果保存为文件，每天固定时间运行一次，以保持代理IP的新鲜性。



**图1 IP代理获取示意图**

工程目录保存在：project/app/kg/my\_proxy\_pool;

程序project/app/kg/my\_proxy\_pool/proxy\_getter/getFreeProxy.py为代理IP获取的主要程序;程序/data/lichengDownload/project/app/kg/my\_proxy\_pool/schedule/timeRun.s为每天固定时间运行程序的脚本;

代理IP文件路径为：project/app/kg/my\_proxy\_pool/schedule/validIp。

1. **对抓取网页进行解析**

解析程序路径为project/app/kg/extract/extract.py，提取了疾病部位与疾病名称，结果在目录：project/app/kg/extract/disease\_location下；提取了疾病与科室，结果在路径project/app/kg/extract/extract.txt。

**二、自然语言处理部分**

自然语言处理是人工智能与语言学领域的分支科学，是指计算机对言语信息进行处理的方法和技术。这部分我主要做了三个工作：统计常见主谓搭配、姓名搜索、疾病名称相似度匹配(未完成)。

1. **统计常见主谓搭配**

统计主语和谓语的常见搭配，用来得到一些常见的症状。

主语包括：organ、tissue、indicator、function、nutrition；谓语包括：problem、appearance。

最后统计出5\*2\*2 = 20个词典，包括主谓搭配、谓主搭配共20组词典。

有两种方案可以选择使用：a.使用ahocorasick算法对主诉进行多模式匹配，提取出主谓词，根据主谓词距离决定该主谓词是否为主谓搭配；b.对一段主诉先用LTP进行分词，判断哪个词是主语词，哪个是谓语词，再用词与词之间的依存关系决定是否为主谓搭配。

1. **基于ac算法的常见主谓搭配**

该部分的测试结果效果并不好，噪声很大，原因是通过距离来判断主语词和谓语词是否是主谓搭配非常不准确。

1. **基于LTP分词的常见主谓搭配**

该部分代码的路径为：/data/user/licheng/project/app/hospital\_guide\_robot/backends/nlp\_licheng/sub\_pre\_common/based\_ltp/sub\_pre\_com\_latest.py。测试效果不错。最后的结果在目录：/data/user/licheng/project/app/hospital\_guide\_robot/backends/nlp\_licheng/sub\_pre\_common/based\_ltp/res\_sub\_pre 下。

1. **姓名搜索**

用户用语音输入一个姓名，通过百度/科大讯飞语音转汉字接口将姓名语音转化为汉字字符串，然后输入到姓名**搜索模块**匹配出若干相似的姓名。自定义一个打分规则用于匹配的姓名打分。

测试分为三项，形状、拼音、音调，总分200分。

第一项：形状得分，满分50分，得分初始值score = 0。

* 若输入姓名与匹配的姓名长度不一样，则不得分，返回分数；
* 输入姓名的长度为*L*，与匹配的姓名对应位置有*x*个相同的词，则得分score += 50 / *L \* x*，返回分数。

第二项：拼音得分，满分100分，得分初始值score = 0。

* 若输入姓名拼音的长度与匹配的姓名的拼音长度不一样，则不得分，返回分数；
* 输入姓名拼音的长度为*L*，与匹配的姓名对应位置有*x*个相同的拼音，则得分score += 100 / *L \* x*；
* 输入姓名拼音有*y*个拼音与匹配的姓名对应位置的拼音结尾仅差一个*g*或者开头仅是*l*和*n*的不同，则得分score += ( 100 / *L –* 10 ) *\* y*；
* 输入姓名拼音有*z*个拼音与匹配的姓名对应位置的拼音结尾仅差一个*g*且开头仅是*l*和*n*的不同时，得分score += ( 100 / *L –* 15 ) *\* z*；
* 若输入姓名有*w*个拼音与匹配的姓名对应位置的拼音不同，但是首字母相同，，则得分score += 5，返回分数。

第三项：音调得分，满分50分，得分初始值score = 0。

* 若输入姓名的音调与匹配的姓名的音调长度不一样，则不得分，返回分数；
* 输入姓名音调的长度为*L*，与匹配的姓名对应位置有*x*和形同的音调，则得分score += 50 / *L* \* *x*，返回分数。

最后将三项加在一起算一个总分，再除以200，这样最大的值为1。

程序目录在：project/app/hospital\_guide\_robot/backends/nlp\_licheng/sub\_pre\_common/based\_ltp。其中，name\_search.py为主要程序；先将姓名放入name\_dict中，运行file\_name2name\_pinyin.py会产生一个中间文件name\_yin\_tone\_dict，姓名搜索的程序：name\_search.py文件中NameSearch对象的run函数输入为汉字名字，输出为匹配名字的list，list的各项为(‘name’,score)。

1. **疾病名称相似度匹配**

一个疾病名称有不同的说法，比如“肚子疼”，可以叫成“腹部痛”，“腹疼”，“肚子痛”等，关于疾病有很多种，有很多疾病名称，如果手动将每种疾病的名称分到相关疾病的目录下，那么工作量将十分巨大，所以需要一个自动将相关疾病名称分到相关疾病的下面。

思路为统计一个疾病名称与对应的其他疾病名称共现的频率，构建疾病的向量；然后计算疾病名称向量的距离，将距离短的聚类在一起。

程序在project/app/hospital\_guide\_robot/backends/nlp\_licheng/disease\_match/match.py，目前完成了共现词频率词典的构建，以及词典的反转，相似度计算尚未完成。

在京东方实习这段时间，我学到了很多东西，感觉非常充实，非常感谢李老师的悉心指导，李老师学识渊博，是我学习的榜样。