目录

1 实验介绍

1.1 实验内容

本次报告包括了第四次和第五次上机实验的内容, 如下:

1.1.1 实验四

实现一个中文网页索引与搜索程序, 爬取一定数量 (>5k) 的中文网页 (可利用之前实验爬取的网页), 修改 IndexFiles.py 和 SearchFiles.py, 对这些中文网页建立索引并进行搜索, 搜索时需要打印出检出文档的路径、网页标题、url 等.

即 doc 的 Field 中需要有 name(文件名),path(文件路径),title(网页标题),url(网页地址),contents(索引的文件内容).

1.1.2 实验五

1. 模拟实现搜索引擎的"site:" 功能 (对搜索的网站进行限制), 提示: 可以在原先的索引上添加一个可以索引的 domain 域, 来对网址所在的域名进行索引.

2. 实现一个图片索引: 新建一个索引, 输入文本, 输出相关的图片地址, 图片所在网页的网址, 图片所在网页的标题等. (做图片索引时最好选定某个你感兴趣的网站爬取, 比如只对糗事百科上的图片进行索引, 这样可以对特定网站的结构进行分析, 让搜索结果更精确.)

1.2 实验环境

操作系统: Ubuntu 18.04 LTS
Python 版本: 3.7
PyLucene 版本: 7.4.0

1.3 所需技能

实验四主要涉及 PyLucene 的安装以及基本使用方法, 需要对 Linux 环境下软件的编译与安装有所了解, 同时也需要理解信息检索的一些基本概念, 譬如中文分词与倒排索引.

实验五对 PyLucene 进一步深入学习, 实现多条件检索以及图片检索, 需要对 PyLucene 的工作机制有更多的认识.

2 实验过程

2.1 准备实验环境

实验指引提供了两种安装 PyLucene 的方法: 通过 conda 安装 PyLucene 4.10.0 或者手工编译安装 PyLucene. 考虑到经过多年的迭代,PyLucene 最新版本已然是 7.4.0, 和 conda 源相距了 3 个主版本, 不免觉得后者陈旧. 于是, 我决定手工编译安装最新版本的 PyLucene.

然而, 关于编译安装 PyLucene 的方法, 实验指引所提供的方法并不十分有效 (按照指引多次尝试均告以失败). 参照 PyLucene 官方提供的安装方法 [1] , 以及摸索尝试, 最终我找到了完全可行且同时适用于 Python 2 和 Python 3 的方法, 如下:

2.1.1 安装 JDK 和 Ant

实验指引给出的命令是: `sudo apt-get install default-jdk` , 但经过实验, 由于编译 JCC 的主文件 setup.py 的缺陷, 使用 open-jdk 会存在编译失败的可能. 在粗略查看了 setup.py 的源代码后, 我发现其默认使用的 JDK 是 Oracle JDK 8(这一点似乎甚至写死在代码里了), 那么, 与其费力修改, 不如按照要求, 安装 Oracle JDK 8. 至于如何安装, 大致过程是从官网下载后手动安装, 在此不表, 需要注意的是, 安装后需要将 JDK 路径等加入环境变量, 同时设置系统的默认 JDK [2] (如果安装了多个 JDK).

至于安装 Ant, 则较为简单, 只需 `sudo apt install ant` .

2.1.2 编译安装 JCC

实验指引给出的命令是: `sudo easy_install jcc` , 但这样的方法并不奏效, 并且无法为系统中不同的 Python 安装 JCC. 因此, 更好的办法是手动编译安装.

在从 PyLucene 官网下载了 PyLucene 源文件后, 解压, 进入 `./jcc/` 文件夹内, 根据需要安装 JCC 的 Python, 尝试运行: (其中的 python3 可换成 python2, 抑或系统内其他的 Python)

```
python3 setup.py build
sudo python3 setup.py install
```

但这么做存在失败的可能,setup.py 可能无法正确的检测到系统内的 JDK, 因此, 如果失败, 需编辑 setup.py 文件, 在约 70 行附近, 根据实际的 JDK 路径, 修改如下内容后再次尝试:

```
JDK = {
    'darwin': JAVAHOME or JAVAFRAMEWORKS,
    'ipod': '/usr/include/gcc',
    'linux': '/usr/lib/jvm/java-8-oracle', #edit this line!
    'sunos5': '/usr/jdk/instances/jdk1.6.0',
    'win32': JAVAHOME,
    'mingw32': JAVAHOME,
```

[1] https://lucene.apache.org/pylucene/install.html

[2] 设置方法可参照 https://www.digitalocean.com/community/tutorials/how-to-install-java-with-apt-on-ubuntu-18-04#setting-the-java_home-environment-variable

```
    'freebsd7': '/usr/local/diablo-jdk1.6.0'
}
```

2.1.3 编译安装 PyLucene

回到 PyLucene 的主文件夹内, 修改 Makefile 文件, 无需理会其中的系统信息, 直接在文件开始添加如下几行:

```
ANT=ant
PYTHON=python3
JCC=$(PYTHON) -m jcc
NUM_FILES=8
```

当然, 这么做的前提是在正确的设置了环境变量. 而后编译安装即可:

```
make
make install
```

2.2 爬取数据

2.2.1 准备工作

在前几次的实验中, 我们分别实现了基本的 crawler 以及 parser, 并为其实现了诸如多线程,BloomFilter, 自动检测编码等功能. 为了方便后续的使用, 我对原有的代码进行了重构与整合, 作为工具类, 封装为一个 Python Package. 其结构如下:

```
my_utils
    __init__.py
    bloom_filter.py
        class Bitarray()
        class BloomFilter()
    parsers.py
        class BaseParser()
        class QSBKParser()
    crawlers.py
        class BaseCrawler()
        class MultiThreadingCrawler()
```

其中改动较大的部分为`MultiThreadingCrawler()`: 增加了 Session 池, 哈希文件名等功能.

Session 池的目的在于提高爬取的效率, 无需为每次请求生成单独的 Session, 同时也对被爬取网站更加友好, 避免同时产生多个连接, 从而降低其负担. 采用的实现方式较为简易: 初始化时生成一定数量的 Session 并放入池中, 每次请求时随机从池中取出一个 Session 使用, 代码如下:

```python
def __init__(self, thread_num=4, headers=None, session_num=10,
             index_file=None, data_folder=None, debug=False, verbose=True):
    # ...
    self.sessions = list()
    for i in range(session_num):
        self.sessions.append(requests.Session())
    # ...

def get_html(self, url, headers=None):
    assert len(self.sessions) != 0, "There's no Session available!"
    current_session = random.choice(self.sessions)
    raw_html = current_session.get(url, headers=headers).content
    # ...
```

哈希文件名的目的在于规范保存文件时的文件名, 避免出现特别长的文件名 (爬取过程中会遇到某些非常长的 URL). 特别长的文件名可能会遇到系统层面的不支持, 并且在后续的实验中,PyLucene 创建索引时遇到长文件名会报错. 因此, 可以将 URL 经过哈希处理后再作为文件名, 则可保证文件名的规范性:

```python
def __hash_filename(self, s):
    md5 = hashlib.md5()
    md5.update(s.encode("utf8"))
    return md5.hexdigest()+".html"
```

2.2.2 爬取 HTML 数据

根据实验要求, 需要爬取至少 5000 个网页并存储在本地, 为了提高爬取的效率, 使用多线程爬取器, 即 `MultiThreadingCrawler()` 进行较为合适. 然而出于爬虫的礼貌性, 过于高频地访问同一主机又是不合适的. 因此, 需要为每次请求设定一定的的时间间隔.

此外, 为了爬取到类别丰富的网页, 同时也为了避免给同一网站造成过大的访问压力, 可以设定爬取起点为导航类网站, 如 hao123. 从一定意义上思考, 这相当于对整个互联网进行了取样, 保证了爬取图的连通性.

结合以上几点, 爬取 HTML 数据的脚本 (crawl_data.py) 如下:

```python
from my_utils.crawlers import MultiThreadingCrawler
import sys
import os

if __name__ == '__main__':
    seed = "https://www.hao123.com/"
    max_page = 5000
    thread_num = 16
```

```
    sleeping = 5
    session_num = 32
    data_dir = "data"
    if not os.path.exists(data_dir):
        os.mkdir(data_dir)
    index_file = os.path.join(data_dir, "index.txt")
    data_folder = os.path.join(data_dir, "html_data")
    headers = {
        # omitted
    }
    my_crawler = MultiThreadingCrawler(thread_num, headers, session_num,
                                       index_file, data_folder, debug=False,
                                       ↪  verbose=False)
    my_crawler.crawl_from(seed, max_page, thread_num, sleeping)
```

几十分钟之后, 爬取完成.

2.2.3 爬取图片数据

爬取图片本身并不困难, 只需提取出网页中的图片元素即可, 然而出于检索的目的, 需要对图片的内容有所” 了解”, 这就需要从图片所在网页提取相关的描述性信息. 考虑到不同网页结构的差异, 在本次实验中, 我仅仅针对单一网站进行图片检索. 利用之前实验中实现的糗事百科解析器, 可以比较方便地完成爬取 (crawl_pic.py):

```
import os
from my_utils.crawlers import BaseCrawler
from my_utils.parsers import QSBKParser

if __name__ == '__main__':
    next_page = "https://www.qiushibaike.com/pic/"
    data_dir = "pic_data"
    if not os.path.exists(data_dir):
        os.mkdir(data_dir)
    index_file = os.path.join(data_dir, "index.txt")
    data_folder = os.path.join(data_dir, "html_data")
    headers = {
        # omitted
    }
    crawler = BaseCrawler(headers=headers)
    parser = QSBKParser()
    while next_page:
        print(next_page)
        html_content = crawler.get_html(next_page)
```

```
        docs, next_page = parser.parse_page(html_content,
        ↪  current_url=next_page)
        with open(index_file, mode="a", encoding="utf8") as index:
            for post in docs.values():
                index.write("{img}\t{content}\n".format(img=post["img_url"],
                    content=post["content"]))
    print("Done!")
```

2.3 建立索引

2.3.1 提取 HTML 文档内容

实验指引中给出了两种方法: 使用 BeautifulSoup 或 nltk 库. 但经过实验, 后者无法正常使用, 而前者的效果并不理想. 以百度首页为例, 其提取的文本为:

```
html \n STATUS OK    百度一下，你就知道
新闻    hao123    地图    视频    贴吧      登录
document.write(\'<a href="http://www.baidu.com/bdorz/login.gif?login&tpl=
mn&u=\'+ encodeURIComponent(window.location.href+ (window.location.searc
h === "" ? "?" : "&")+ "bdorz_come=1")+ \'" name="tj_login" class="lb">
登录 </a>\');\r\n    更多产品                关于百度    About Baidu
©2017\xa0Baidu\xa0 使用百度前必读 \xa0  意见反馈
\xa0 京 ICP 证 030173 号\xa0                \n
```

可以发现, 里面夹杂了不少非文本内容. 因此, 我使用了一个更加优秀的 Python 库:html2text, 用法如下:

```
from html2text import HTML2Text
h = HTML2Text()
h.ignore_links = True
h.ignore_images = True
h.handle(raw_html)
```

同样以百度首页为例, 经过处理后, 效果明显优于 BeautifulSoup:

```
\n\n 新闻 hao123 地图 视频 贴吧 登录 更多产品\n\n 关于百度 About Baidu
\n\n(C)2017 Baidu 使用百度前必 读  意见反馈 京 ICP 证 030173 号\n\n
```

2.3.2 中文分词

由于 PyLucene 本身对于中文无法正确分词, 我们需要借助其他工具来对中文分词, 然后再交由 PyLucene 建立索引. 我使用 Jieba, 一个方便易用的 Python 库, 来进行分词. 考虑到建立索引的需要, 在分词时需要进行一些额外的处理: 去除停用词, 长词切分. 代码如下:

```python
class IndexFiles(object):

    def __init__(self, root, storeDir, analyzer, type="html"):
        # ...
        self.load_stop_words(["CNstopwords.txt", "ENstopwords.txt", ])
        # ...

    def index_html(self, root, writer):

        # ...
        content = jieba.cut_for_search(content)
        content = " ".join(word for word in content
                if word.strip() and word not in self.stop_words)
        # ...

    def load_stop_words(self, file_list):
        self.stop_words = set()
        for file in file_list:
            print("Loading stop words from", file)
            try:
                count = 0
                with open(file, mode="r", encoding="utf8") as f:
                    for line in f:
                        word = line.strip()
                        if word not in self.stop_words:
                            self.stop_words.add(word)
                            count += 1
                        print("\r", count, sep="", end="")
                print("\r{0} word(s) added.".format(count))
            except Exception as e:
                print("Error!")
                print(e)
```

2.3.3 网页索引

对于一个网页, 根据实验四和五的要求, 需要对其内容, 标题, 网址, 存储文件名, 存储路径以及主机进行索引. 关于 content, 在上一小节已经叙述了如何处理; 网址, 存储文件名, 存储路径都可直接从 index.txt 中读取; 而标题, 可以使用正则表达式从 HTML 文档中匹配, 也可以用 BeautifulSoup 解析网页寻找 title 标签即可; 而主机名, 简单的做法是使用 urllib.parse 里的 urlsplit, 将 url 分划后取 netloc 值. 如下:

```
url, path = line.strip().split("\t")
name = os.path.split(path)[-1]
with open(path, mode="r", encoding="utf8") as file:
    content = file.read()
soup = BeautifulSoup(content, "html.parser")
title = soup.title.text if soup.title else "No Title!"
site = urlsplit(url).netloc
```

明确了如何对一个网页进行索引后, 只需对 index.txt 中每一行进行处理, 读取 url 和文件路径, 并加入到索引中, 这一部分参照 PyLucene 提供的实例代码[3]即可, 在此不表.

2.3.4 图片索引

对图片进行索引的方法与网页索引大致相同, 在具体步骤上甚至更为简单.

对于糗事百科的图片, 需要索引的信息有: 图片 url, 图片对应的文字描述, 图片所在网页 url 等. 由于在创建索引时对文字描述进行了分词处理, 为了在搜索时能够显示出正确的文字描述, 需要额外保存一份原始的文字描述. 如下:

```
doc = Document()
doc.add(Field("raw_content", content, t1))
content = " ".join(word for word in jieba.cut_for_search(content)
    if word.strip() and word not in self.stop_words)
doc.add(Field("url", image_url, t1))
doc.add(Field("content", content, t2))
writer.addDocument(doc)
```

2.4 执行搜索

2.4.1 网页搜索

在对网页进行搜索时, 主要是根据 content 键中的词频, 但根据实验五的要求, 还需要支持对 site 键进行限定. 因此, 需要使用 `BooleanQuery()` 实现多条件搜索的功能.

在获取了用户原始输入之后, 首先需要从中解析出不同的键值对:

```
def parse_command(command):
    allowed_opts = ["site", ]
    command_dict = dict()
    opt = "content"
    for s in command.split():
        if ":" in s:
            opt, value = s.split(":")[:2]
            opt = opt.lower()
```

[3] http://svn.apache.org/viewvc/lucene/pylucene/trunk/samples/IndexFiles.py

```
            if opt in allowed_opts and value:
                command_dict[opt] = value
        else:
            command_dict[opt] = command_dict.get(opt, "") + " " + s
    return command_dict
```

然后根据返回的命令字典, 生成多个查询子句, 添加到`BooleanQuery.Builder()`中, 最后以此构造出完整的布尔查询. 需要注意的是, 对于 content 的搜索, 仍需要对其进行分词处理, 以保持和创建索引的一致性. 主要代码如下:

```
command_dict = parse_command(command)
querys = BooleanQuery.Builder()
for k, v in command_dict.items():
    if k == "content":
        cutted = [x for x in jieba.cut_for_search(v) if x.strip()]
        v = " ".join(cutted)
    query = QueryParser(k, analyzer).parse(v)
    querys.add(query, BooleanClause.Occur.MUST)
querys = querys.build()
```

2.4.2 图片搜索

对图片的搜索可以暂时不考虑 site 限制 (目前仅爬取了单一网站的图片), 那么, 事情就变得十分简单了. 直接贴出图片搜索的完整代码, 如下:

```
def search_image(searcher, analyzer):
    while True:
        print("Hit enter with no input to quit.")
        command = input("Query:")
        os.system("clear")
        if command == "":
            return
        print("Searching for:", command)
        command = " ".join(x for x in jieba.cut_for_search(command))
        query = QueryParser("content", analyzer).parse(command)
        scoreDocs = searcher.search(query, 10).scoreDocs
        print("{} total matching documents.".format(len(scoreDocs)))
        for num, scoreDoc in enumerate(scoreDocs):
            doc = searcher.doc(scoreDoc.doc)
            print("\n#{num}:\nURL:{url}\nContent:{content}\n".format(
                num=num + 1, url=doc.get("url"),
                ↪  content=doc.get("raw_content")))
```

3 总结与分析

坦白地说, 本次实验是令人沮丧的——我的大部分时间花在了编译安装 PyLucene 上. 实验伊始, 我试着按照指引手动编译安装 PyLucene, 但多次尝试均无果, 不得不自行查找解决方案, 其间查阅了 PyLucene 的官方教程, 也粗看了其编译文件的源代码, 最终方能成功安装.

不过尽管如此, 这次实验中我将以往实验所写代码进行了重构和整合, 再次感受到了面对对象的高效与某些 Python 独特语法的妙用. 至于 PyLucene, 我学到了其基本用法, 并实现了一个简易的搜索引擎, 这一过程也颇有收获.