上海交通大學

SHANGHAI JIAO TONG UNIVERSITY

电类工程导论 (C 类) 终期报告

-2020 Fall EE208 Group 5-

2021 年 1 月 12 日

体育类新闻的数据爬取和搜索引擎的搭建

宋家鑫 陈安瑞 祝福泽

2021 年 1 月 12 日

摘要

在本次项目中，我们先利用并行式爬虫爬取 1w 余体育网页和 6k 余张包含大量人脸的图像。然后对这些新闻数据进行解析，并建立索引。在搜索引擎的创立过程中，我们利用 lucene 进行关键词查询。然后通过 ResNet50 模型提取图像特征信息，并基于 LSH 算法进行优化搜索。整个网页采用 Flask 库进行搭建，并借助 bootstrap 框架优化页面设计。

bonus: 实现了基于 k-means 的新闻聚类和基于人脸的新闻搜索，并在网页设计上加入了基于遗传算法的 AI 绘画。

1 依赖的库和运行环境

依赖的库和运行环境：

- * opencv 3.4.1 * numpy 1.16.2 * matplotlib 3.0.3 * jieba 0.42.1 * Flask 1.1.2 * dlib 19.21.1
- 本次实验在 docker 环境下执行

2 爬虫搜集原始数据

2.1 具体流程

我们小组在进行网页爬取的时候采用的是 DFS 方法。[1]我们尝试爬取的网页有：新浪体育、网易体育、腾讯体育、新华体育、体育新闻网、搜狐体育、人民体育网。

其中，在爬取新浪体育、网易体育、腾讯体育的过程中发现其中的无关网页比例非常高，严重影响爬取速度，故只爬取了少量这些网页，而人民体育网的新闻页面中很少有图片，且爬取后得到的内容出现了很多乱码，故舍弃这些数据。最后发现新华体育、体育新闻网、搜狐体育这三个网站的网页质量和有效网页比例都很高，所以将其作为主要的数据来源。

在获取链接域名后，使用正则表达式来判断其格式是否符合预期的格式（预期格式与具体爬取的网页有关，需要爬取多个网页后总结其规律），举例来说，新华网体育中的有效体育新闻页面基

[1]**Note:** 即给定一个初始网页链接作为 seed，存储网页 html 文档中包含的链接，再对这些链接调用相同的搜索函数，直到不再获得新的链接地址或者达到爬取网页数。

本都遵循“http://sports.xinhuanet.com/c/+ 年月日”的格式。若爬取到的链接地址遵循预设的格式则将其保存到 index.txt 文件中，并保存网页 html，否则跳过。

2.2 爬虫过程中遇到的问题

- 开始时我们组选的主题是购物网站，于是尝试对当当网上的图书页面进行爬取，期间我们实现了对图书商品基本信息（静态页面）的爬取，还通过查找资料实现了对网页中的评论信息以及好评度的爬取功能（ajax 传参）。但在实际进行爬取的过程中，一方面没有评论的商品页面较多，得到的信息量较少，另一方面我们的程序经常被当当网 kill，导致爬取的速度非常缓慢，于是我们将主题换为更容易爬取数据的新闻网站。
- 最初我们组直接使用 lab2 中的爬虫代码爬取数据，但发现如果仅仅使用深度优先搜索的话，得到的结果中无效数据（无关的网站）过多，于是我们在储存链接地址前使用了正则表达式匹配的方法进行一定的筛选，爬取结果有了明显的改善。
- 爬取到的网页 html 文本中常出现二进制格式的内容，甚至可能有出现乱码的情况，对于大部分上述情况，我们在采用 decode 函数进行解码以后都可以得到正确的中文字内容，而对于无法处理的部分，我们进行适当的舍弃。

3 新闻聚类

该部分使用 K-means 库对网页标题进行聚类操作，针对一个具体的网页按照相关性来进行网页推荐。

3.1 算法原理：

我们使用 K-means 算法来进行聚类，K-means 算法是机器学习中较常用的算法，属于无监督学习算法，其具体过程如下：

Algorithm $K-means$
step1：预设分类数量 k
step2：适当选择 k 个类的初始中心
step3：在第 i 次迭代中，对任意一个样本，求其到 k 个中心的距离，将该样本归到距离最短的中心所在的类;
step4：利用均值等方法更新该类的中心值;
step5：迭代上述过程，直到利用（3）（4）的迭代法更新后中心值保持不变

3.2 核心代码及其说明

代码 1：预处理网页标题，并调用 jieba 进行中文分词。

代码 2:

图 1: 计算 TF-IDF:

```
def transform(dataset,n_features=1000):
    vectorizer = TfidfVectorizer(max_df=0.5, max_features=n_features, min_df=2,use_idf=True)
    tmp=[]
    for i in dataset:
        tmp.append(i[0])
    X = vectorizer.fit_transform(tmp)
    return X,vectorizer
```

代码 3:

说明： 通过调用 TfidfVectorizer 文本特征提取器来计算 TF-IDF（词频-逆向文件频率），并建立矩阵，为后续使用 K-means 算法做准备。

代码 3:

图 2: 训练模型:

```
def train(X,vectorizer,true_k=15,minibatch = False,showLable = False):
    #使用采样数据还是原始数据训练k-means,
    if minibatch:
        km = MiniBatchKMeans(n_clusters=true_k, init='k-means++', n_init=1,init_size=1000, batch_size=1000, verbose=False)
    else:
        km = KMeans(n_clusters=true_k, init='k-means++', max_iter=300, n_init=1,verbose=False)
    km.fit(X)
    if showLable:
        print("Top terms per cluster:")
        order_centroids = km.cluster_centers_.argsort()[:, ::-1]
        terms = vectorizer.get_feature_names()
        print (vectorizer.get_stop_words())
        for i in range(true_k):
            print("Cluster %d:" % i, end='')
            for ind in order_centroids[i, :10]:
                print(' %s' % terms[ind], end='')
            print()
    result = list(km.predict(X))
    print ('Cluster distribution:')
    print (dict([(i, result.count(i)) for i in result]))
    return -km.score(X),result
```

说明： 基于爬取到的网页数据，使用 K-means 算法来训练模型。在模型训练完毕后，将得到的分类结果中每一类的代表内容进行展示，并打印类别分布情况。

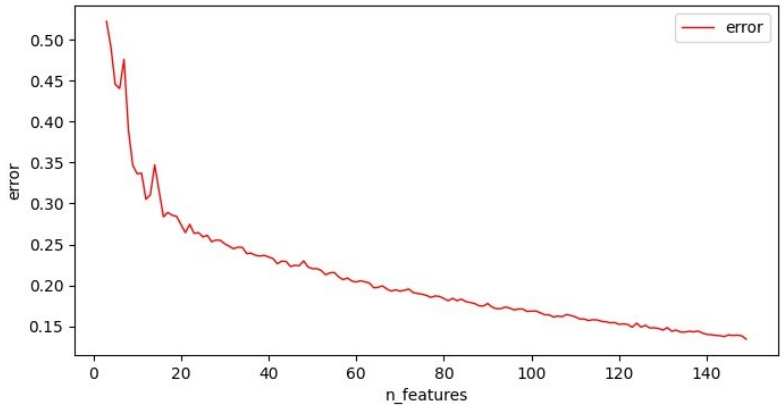
代码 4:

图 3: 测试:

```python
def test():
    '''测试选择最优参数'''
    dataset = loadDataset()
    print("%d documents" % len(dataset))
    X,vectorizer = transform(dataset,n_features=500)
    true_ks = []
    scores = []
    for i in range(3,150,1):
        score,rs = train(X,vectorizer,true_k=i)
        score=score/len(dataset)
        print (i,score)
        true_ks.append(i)
        scores.append(score)
    plt.figure(figsize=(8,4))
    plt.plot(true_ks,scores,label="error",color="red",linewidth=1)
    plt.xlabel("n_features")
    plt.ylabel("error")
    plt.legend()
    plt.savefig('result.jpg')
    plt.show()
```

说明： 由于 K-means 算法需要预先规定聚类类别的数量，而这很难主观地去确定，于是先通过对一个范围内的类别数进行测试来获得较好的数值。测试结果如下图所示:

图 4: 测试结果:



观察曲线可以看到第一个拐点出现在 15 左右，于是我们取 n_features 为 15。

图 5: 测试:

```
import jieba.analyse as analyse
file=open('newdict.txt','r')
data=file.read()
data=eval(data)
result={}
for i in range(len(data)):
    tmp=data[i]
    word=''
    for j in tmp:
        word=word+tmp[0]
    keywords =(analyse.extract_tags(word , topK=5, withWeight=False, allowPOS=(['ns', 'n', 'vn', 'v'])))
    result[i]=keywords
```

说明： 使用 jieba 分词器对每个分类中的标题文本进行关键词统计，这里选择关键词中词频最高的 5 个。

3.3 聚类实现过程中遇到的问题

- 开始时分类结果非常不均匀，在 0 和 1 两类中集中了大约 90% 的数据，在使用 test 函数观察曲线并修正分类数量后，分类情况有了明显的改善。
- 第一次运行程序后分类结果中的网页数量仅有 4000 个，明显小于 12000 的总数据量。检查代码发现有些 html 文档中没有 head 节点，title 节点是独立存在的，于是这些网页被跳过了。将节点定位方式改为直接找到 title 节点后结果变正常了。

4 新闻索引的创建

4.1 html 内容解析

该部分的源代码封装在 parser.py 文件中的 parser() 类中。在调用过程中先初始化 url 和 filename 对应的字典。

观察新闻网对应的 html 结构，其中 title 和 date 都可以通过简单的字符串匹配方式进行提取。而对于 tips 和 related_news，他们存放具有特定类型的类的 div 中，需要先将 html 文本解析成语法树，然后再寻找相应的标签，并进行提取。

4.2 索引创建

该部分代码与之前 lab4 的内容相近，源代码放在 IndexFiles.py 文件夹下。程序首先遍历 html 文件夹，并查找其 url_filename() 词典，寻找对应的 url。并调用 parser.py 中的函数解析 html 文本，并将同一条新闻的各项存在同一个 doc 中。

5 图片库的创建和利用 LSH 优化的图片搜索

5.1 图片库的建立

遍历爬取到的 html 文件，并存储每个 img 标签对应的链接的图片，我们得到了大小约 6000 的图片库。为了方便查询到图片后对应相应的图片，我们将图片命名为：

filename:html 文件名 +'%'+str(i: 网页中第 i 张图片)+'.jpg'

5.2 人脸图片库的建立

5.2.1 环境准备

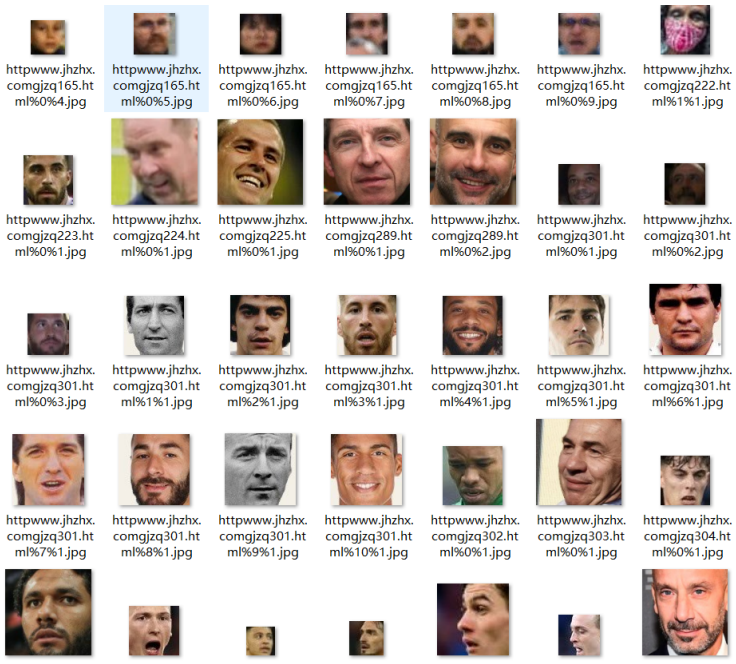
我们该部分实现依赖于 dlib 库。Dlib 库是一个包含了机器学习算法的 C++ 开源工具包，但使用 C++ 编写。因此我们并不能直接通过 pip 安装，而需要先 pip 预安装 Cmake 和 boost 库。

5.2.2 建立人脸图片库

建立人脸图片库的源代码被封装在 extract_faces.py 文件下。对于目录下每张图片，调用 dlib 库下的 get_frontal_face_detecto() 函数，截取其人脸边框部分。并依据人脸边框的大小，新建一张空的图片并将人脸写入。

图片命名规则为：filename：html 文件名 +%i: 该网页下的第 i 张图片 +% 该图片中第 i 张人脸 +'.jpg'。遍历图片库，得到人脸图片库：

图 6: faces_database:



5.3 图片特征信息的提取

特征信息提取的代码放在 extract_features.py 文件中。之前实验中 SIFT 算法提取特征信息速率较慢，且准确度不高。本次实验采取 resnet50 模型进行图片特征信息提取。

Algorithm:*Extract_features*
step1 : 卷积操作
step2 :4 个残差阶段
step3 : 均值池化

5.4 基于 LSH 的高维空间搜索:

该部分源代码封装在 lsh.py 文件下。算法流程如下:

Algorithm:LSH
step1 : 定义 L 个哈希函数，每个哈希函数有 k 个取值
step2 : 初始化一张 L×k 的哈希表
step3 : 遍历每个向量，并计算其 L 个哈希函数值，并将其存入哈希表中

在实验中，我们取 L=32，定义哈希函数为 $f_i(x) = (x[64i+1] \geq 1) * 1 + (x[64i+33] == 2) * 2$

5.5 基于图片的新闻搜素

前端输入图片后，后端调用 extract_feature.py() 函数生成其特征向量。然后计算其 L 个函数的取值，并进入到哈希表中进行比对，返回相似图片。

根据图片名，可以获取图片对应的 html 文本的 filename，调用 parser 类下的 get_message() 函数，可返回该图片对应的新闻的详细信息。

在前端相似图片处，设置超链接 <a>，链接到'/message' 页面，并用上一步中新闻详细信息实例化模板。

5.6 基于人脸识别的新闻搜索

前端输入图片后，后端首先对其中的人脸进行识别并提取，并调用 extract_features.py 文件生成其特征信息，并利用 LSH 进行搜索，返回相似人脸。然后根据图片名称获取对应的 html 文本的文件名，然后与上一步操作相同，

6 基于 Flask 的前端框架设计

6.1 文件目录结构

-static

-css

-js

-img

-templates

-app.py

Notes: 其中 static 文件夹用来存放包括 css、js 和图片在内的用来渲染网页的静态文件，templates 文件下存放 html 模板。

6.2 网页框架

为了优化网页排版，本次实验选用 bootstrap 作为网页搭建框架，并从 php 中文网上搜集了好看的网页模板。

其中构建的主要网页包括：主页、基于关键词的搜索页面、基于图片的搜索页面、基于人脸的新闻搜索、新闻细节网页。网页的具体演示可以观看 demo 视频。

6.3 网页设计思路

- 文字类搜索：在前端输入文字后，将其传入后端。并用 QueryParser 处理输入的文字信息。用 sportes_index 内存储的索引初始化 analyser 和 searcher 对象。并调用 searchr 寻找相关新闻，并将结果返回给前端。若要求结果依据时间排序，还需要调用 datetime 库，先格式化处理日期，然后将依照时间排序的搜索结果。
- 图像类搜索：在前端输入图片后，将其保存在本地，并将图片名返回给后端。app.py 调用 extract_features.py 提取目标图片的特征信息，并利用 LSH 进行搜索，并将相似图片的路径返回给前端。[2]
- 在实现前后端交互时，我们主要使用 form 表单来从前端获取信息并传递给后端。并在不同的 html 模板中，给 form 不同的 name 属性。

6.4 效果：

- 动态显示特效。
- 鼠标悬浮特效：当鼠标停留在新闻图片上时，会出现悬浮框显示新闻的详细信息。

[2]为了使图片能够在前端加载，本次实验的所有图片都放入 static 文件夹中

- 图片搜索结果可通过 AI 绘画显示

6.5 网页搭建过程中遇到的问题

- 网页无法加载静态 js 和 css 资源：解决此问题，我们需要将 static 文件夹加入到网页的静态资源路径中。具体操作上首先要在 app.py 中添加 $app = Flask(\_\_, static\_url\_path = '/static')$, 然后用 url_for() 函数改写网页模板中调用的静态资源的路径。

- 如何上传本地图片并将其保存在某个目录下？修改 input 输入框的类型为 file，并调整表单发送属性为 multipart/form-data。

7 基于遗传算法的 AI painter

7.1 设计思路:

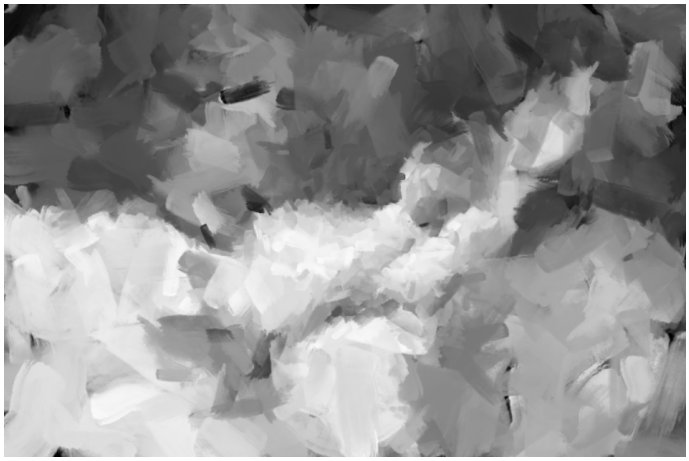
该部分代码封装在 genetic-drawing.py 文件夹下。[3] 其大体思路是，利用遗传算法，将像素值序列充当 DNA 序列。按照原图像的像素值规定了 DNA 演化的方向和环境适应度，将 DNA 的演化过程动态生成一张 gif 图像，模仿绘画过程。

在策略上，先用不带掩膜的方式运行 50 个 stages，每个 stage 下 10 个 generations。然后再按照带掩膜的方式执行 20 个 stages，进行局部精细化处理。

图 7: 搜索结果

[3]设计思路并非原创，来源于 github 上的一个开源项目。我们只是稍微对其进行改进。

图 8: AI 绘图过程



8 效果展示

具体可以观看 github 上 demo 演示视频

9 开源代码

本次课程作业的全部代码已在 github 开源,项目地址:https://github.com/sjxer723/EE208$_final$