基于Scrapy的Python包关系分析

https://github.com/HarryHaoLee/Relationship-Analysis-between-Packages

2018 年 1 月 14 日

目录

1 问题与需求分析

1.1 问题提出

我们将要以python极其各种包、库为主题进行一些分析：分析python的哪些包被使用或提及的频率最高；哪些包之间存在着一些联系；python最常被用于哪些话题；使用python的开源仓库，最经常使用什么协议等。

1.2 思路分析

首先，我们需要获取一些关于python及其包、库的数据。Stack Overflow是一个与程序相关的IT技术问答网站。用户可以在网站免费提交问题，浏览问题，索引相关内容，在创建主页的时候使用简单的HTML。gitHub是一个提供了git代码仓库托管及基本的Web管理界，以及订阅、讨论组、文本渲染、在线文件编辑器、协作图谱（报表）、代码片段分享（Gist）等功能的大型网站。这两个网站包含了大量与Python包相关的信息，所以我们将主要选择github，StackOverflow这两个网站作为数据来源。为了得到每个包被提及或使用的“热度”，我们可以选用它们在tags 中出现的次数，以及在工程代码中被使用的次数；为了获取各个包之间的联系，我们可以统计它们同时出现在一个问题或仓库下以及同时被一个工程所使用的次数。

在具体的数据选择上，我们将分别通过爬虫得到：

- StackOverflow上与python相关的问题的“热度”，每个问题下的各个tag
- github上语言为python的仓库所使用的协议，每个仓库的各个tag
- github上语言为python的仓库，每个仓库中的每个*.py文件中，import的各个包名字

获取数据后，我们将根据每个包自身被提及的“热度”，它们之间同时出现的关联度（每对包每同时出现一次，则认为它们的关联+1），以及其他由爬虫获取的相关信息，对它们进行分析。

在数据可视化上，我们将采用表格、直方图、Gephi绘制网络关系图等的方式，将分析得到的结果进行直观的显示。

1.3 需求分析

功能需求 ：

1. 从指定网站爬取指定数据
2. 通过爬取的数据分析出现频率高的库

3. 通过爬取的数据分析热度高的库
4. 通过爬取的数据分析容易在一起出现的库
5. 通过爬取的数据分析开源软件仓库使用频率高的协议

性能需求 ：

1. 程序规模需限制在普通笔记本电脑可以运行的范围
2. 所需存储空间不应过大
3. 考虑到项目的时间限制，程序的运行时间必须在可接受范围内

可靠性和出错处理需求 ：

1. 爬虫必须有一定的重启机制和对反爬虫机制的预防机制，以防在没有人关注的时候被停止或爬取到大量错误数据。接口需求
2. 程序运行结果必须按照用户需求保存在不同格式的文件如.csv、.txt、.json中，以供保存、阅读或者由其他程序调用。

约束 ：

1. 此程序为对开源软件仓库的爬取分析程序
2. 需使用Python语言完成
3. 需包含数据的获取、清洗、分析、可视化等步骤

将来可能提出的需求 ：

1. 未来可能从更多角度对数据进行分析，所以需要多元化的数据，如Stackoverflow问题中问题的题目、浏览次数、标签等

2 设计

2.1 概要设计

我们的程序在功能上可分为三个模块，数据获取、数据处理、数据可视化，每个模块又可分为若干子模块，较详细的概要设计可参考下图：

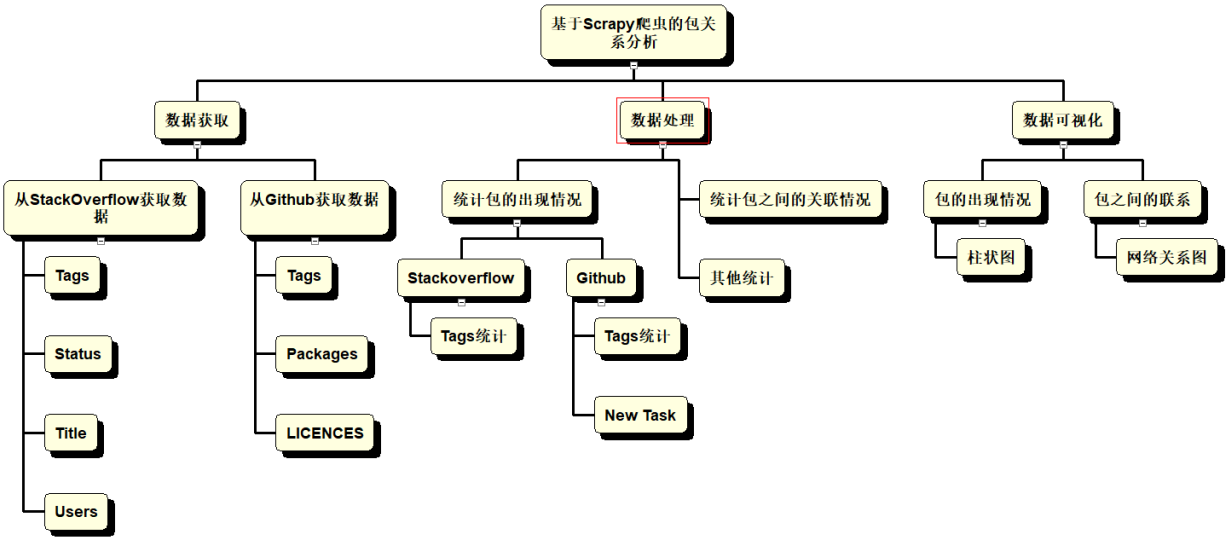

图 1: 层次分析图

2.2 详细设计

使用scrapy对StackOverflow上的问题进行爬取，流程如下：

Algorithm 1 爬取StackOverflow上问题

Require: 待爬取的startUrl

Ensure: 存入MongoDB中的爬取结果

currentUrl = startUrl
while 未达到需要的数量 **do**
    获取currentUrl（为搜索页url，每页有若干个问题摘要）的页面currentPage
    将currentPage交给parse函数
    解析得到每个问题的url: urlList，解析得到下一页的url: nextUrl
    for each url in urlList **do**
        获取页面内容content
        将content交给parse item函数，解析得到页面具体信息info
        将info存入数据库中
    end for
    currentUrl = nextUrl
end while

使用url.request对GitHub上的仓库基本信息以及每个仓库内import的packages进行爬取，流程如下：

Algorithm 2 爬取GitHub上的仓库

Require: GitHub的token，search页面url的组成规则

Ensure: 存入json文件的结果

```
maxStar = inf
while 未达到需要的数量 do
  for page = 1:10 do
    获取搜索页面的url（语言为python，star数小于maxStar，按照star数降序排列，perpage = 100）// GitHub的search API对每次搜索结果仅返回1000 条，因此，为了获得足够的仓库信息，将按照不同的star数范围进行多次搜索
    得到当前页面的内容//为json数据，返回值由字典接收
    for each repository in this page do
      获取该仓库的基本信息basicInfo与仓库内代码内容的url: contentUrl
      packageInfo = getPackage(contentUrl)
      将basicInfo与packageInfo存入json文件中中
      if 当前为第10页最后一条仓库信息 then
        maxStar = 此仓库的star数- 1
      end if
    end for
  end for
end while
```

getPackage的过程如下：

Algorithm 3 getPackage

```
Require: contentUrl
Ensure: packageInfo
contentInfo = []
获取contentUrl页面信息
for each content in this page do
  if Type == file and Name == *.py then
    下载此文件内容并解析得到其中import的所有内容packages
    contentInfo += packages
  end if
  if Type == dict then
    url = 当前目录的contentUrl
    packageInfo += getPackage(contentUrl)
  end if
end for
return packageInfo
```

分析处理数据的流程（对GitHub与StackOverflow上获取的各项数据都可按照相似的流程进行处理，只是对不同数据在一些处理细节上有所不同）如下：

Algorithm 4 getPackage

Require: 爬虫获取的数据文件，较常被使用的包列表packageList

Ensure: 分析统计结果

```
packageCount = {}, packagePopular = {}, packageRelationship = {}, licenseCount = {}
for each record in the database/file（问题或仓库的信息） do
    tags = 此条记录中属于packageList的tag/package
    licenseCount[license] += 1
    for each tag in tags do
        packageCount[tag] += 1
        packagePopular += 此条记录的“热度”
        for each tag2 in tags do
            packageRelationship[(tag, tag2)] += 1
        end for
    end for
end for
return 统计得到的各个结果
```

3 数据获取与过滤处理

3.1 从GitHub获取的数据

按照“详细设计”中的爬虫逻辑，共从GitHub上获取了16800个语言为python的仓库的基本信息用于分析它们的许可证，标签内容。由于下载获取*.py文件较慢，仅爬取了1000个仓库具体代码内容的import packages 内容，用于后续分析。

3.2 从StackOverflow获取的数据

按照“详细设计”中的爬虫逻辑，共从StackOverflow上爬取了36000个问题的热度，标签，用户等信息，用于后续分析。由于部分问题存在数据缺失，例如没有用户回答或者缺少votes数据，我们对这部分残缺的问题进行了删除。

3.3 其他数据

从网络上获取了Python库列表。其中包含Anaconda官网、Python内置库列表、Python第三方库列表等，由于一些数据并没有以清晰的列表的形式给出，我们还设计了一些正则匹配的方法进行获取。最后得到了比较全面的Python常用库列表。

4 数据分析

我们认为一个包出现在Stackoverflow问题中的次数越多，包含它的问题的热度越高，使用它的人数便越多，便可以认为它活跃度较高。同时，若两个包同时出现在一个问题中的次数越高，我们便认为它们的关系越密切。Github中同理。基于这个认知，我们决定统计Stackoverflow中问题出现的次数及与之相关的问题热度，并分别一句出现次数与热度对包排序；统计包共同出现情况并对之进行排序。对于Github亦进行类似统计，由于我们还对Github中1000个工程进行了分析，直接统计了同一工程中包的出现情况，我们认为这部分的数据可信度是较高的。

5 结果可视化与展示

5.1 packages被使用的“热度”结果

在通过爬虫获取的1000个GitHub仓库中，统计它们使用的packages，最常被使用的packages直方图如下：

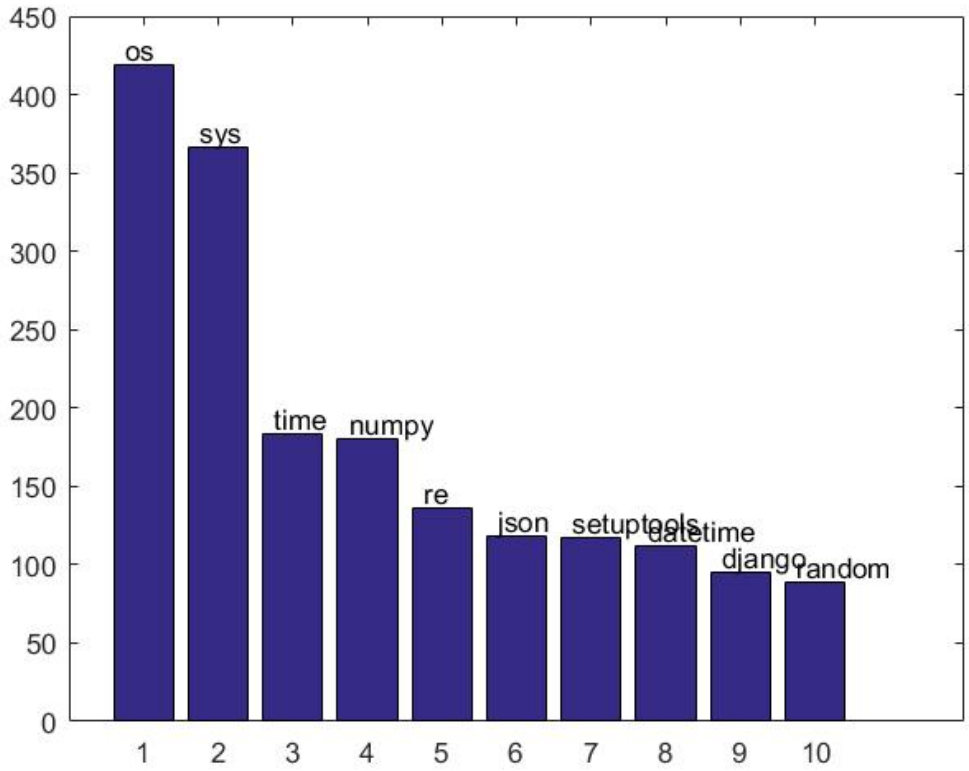

图 2: 统计GitHub上python语言的工程中使用包的情况直方图

他们的详细信息如下:

表 1: 统计GitHub上python语言的工程中使用包的情况

序号	package名	被使用次数
1	os	419
2	sys	366
3	time	183
4	numpy	180
5	re	136
6	json	118
7	setuptools	117
8	datetime	112
9	django	95
10	random	89
11	logging	89
12	matplotlib	83
13	requests	82
14	argparse	78
15	__future__	76
16	math	69
17	subprocess	64
18	pandas	63
19	unittest	59
20	urllib	57
21	collections	53
22	tensorflow	48
23	sklearn	47
24	flask	44
25	distutils	41
26	threading	39
27	scipy	37
28	shutil	36
29	csv	35
30	hashlib	33

统计爬虫获取的16800个仓库中，统计其中提及的packages，最常被提及的packages如下:

表 2: 统计GitHub上python为语言的仓库中tags里提及到包的情况

序号	包名	出现次数
1	tensorflow	330
2	http	42
3	numpy	39
4	json	37
5	sqlalchemy	36
6	pandas	33
7	theano	32
8	jupyter	29
9	scikit-learn	28
10	html	23
11	ipython	21
12	email	20
13	opencv	20
14	matplotlib	20
15	parser	18
16	logging	18
17	requests	17
18	twisted	16
19	csv	16
20	aiohttp	16

统计Stackoverflow36k个问题，最常被提问的packages直方图如下：

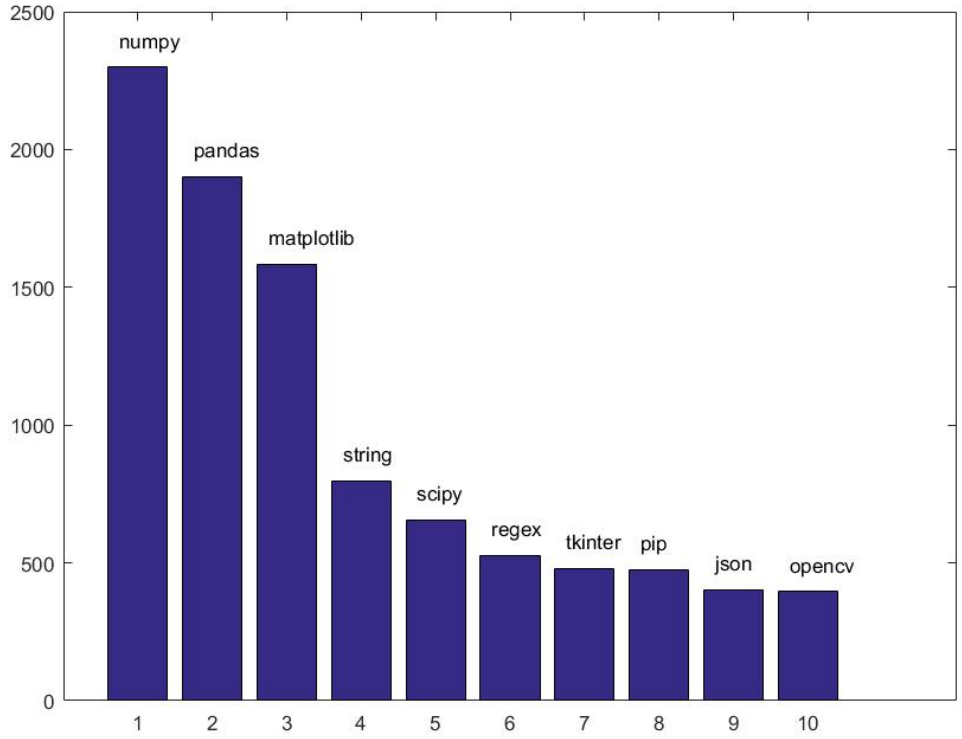

图 3: Stackoverflow上python相关的问题中tags里提及到包的次数直方图

它们的具体信息如下：

表 3: 统计Stackoverflow上python相关的问题中tags里提及到包的情况

序号	包名	出现次数
0	numpy	2229
1	pandas	1902
2	matplotlib	1584
3	string	797
4	scipy	654
5	regex	527
6	tkinter	478
7	pip	474
8	json	401
9	opencv	396
10	csv	386
11	sqlalchemy	378
12	datetime	359
13	subprocess	348
14	multiprocessing	345
15	tensorflow	340
16	virtualenv	297
17	scikit-learn	289
18	pyqt	276
19	ipython	264
20	html	261
21	nltk	223
22	logging	202
23	lxml	174
24	math	170
25	pyqt4	167
26	cython	161
27	pickle	149
28	pygame	142
29	random	139
30	argparse	133

然后我们采用计算votes、favorite_count、answers、views和的方式衡量该问题给该标签添加的热度，并统计每个标签的热度和，可得热度（每个问题的热度和）最高的标签如下：

表 5: 统计Stackoverflow上python为语言的仓库中标签的热度情况

序号	包名	热度
0	string	69064758
1	pandas	45328071
2	numpy	43093597
3	matplotlib	36833422
4	pip	27232593
5	datetime	19288259
6	json	17297712
7	regex	12370782
8	subprocess	10981344
9	csv	10966890
10	scipy	10668052
11	tkinter	8859804
12	virtualenv	8447119
13	time	7351270
14	opencv	6600929
15	types	6246745
16	math	6126282
17	random	6106164
18	ipython	4965214
19	html	4889697
20	logging	4587029

上面的结果显示，在python工程中，一些python自带的包（os, sys, time）最常被使用；第三方库中，numpy被使用频率最高；网页制作框架，数据可视化与数据处理相关的包，机器学习/深度学习相关的包等，出现频率也很高。

在StackOverflow中的提问里，string包有着最高的热度，可以反映出用户在使用该包时遇到的问题是比较多的，数据处理与可视化相关的包（numpy, pandas, matplotlib）被提及的频率均较高，有较高的“热度”。

5.2 packages间的“联系”结果

通过统计得到的信息，我们进行了关系网络图的绘制，起初我们想使用Python中

的matplotlib与Networkx进行绘制，但可能是由于点与边的数量过多，且Networkx 与Python3.x 的兼容性不好，没有办法绘制出图形，所以我们采用了Gephi进行可视化。将stackoverflow上爬取的数据按照['Source','Target','Weight']的形式导入之后，调整参数，可以看到可视化后的图形如下。

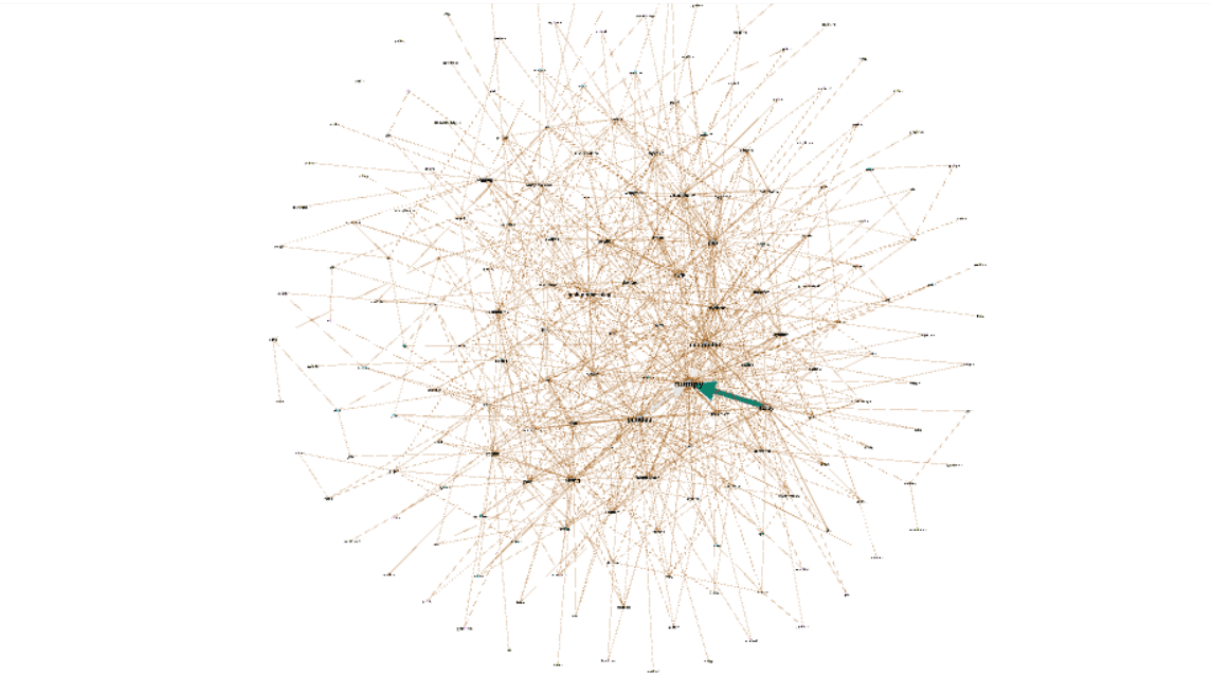

图 4: StackOverFlow问题中的包关系网络图

查看图形的细节如下，边的颜色与尺寸代表了变的权重，点的颜色与尺寸代表了点的度。

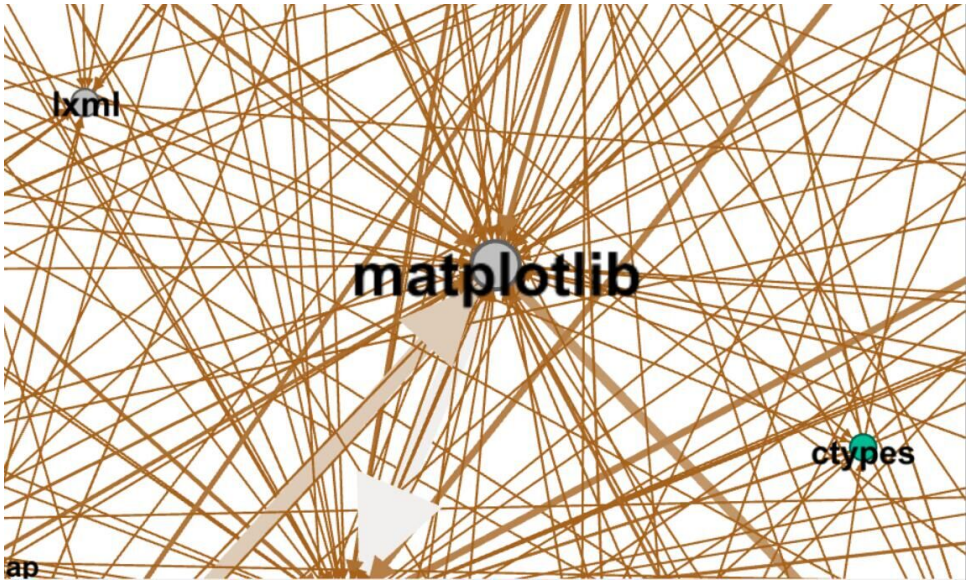

图 5: StackOverFlow问题中的包关系网络图细节

将鼠标放置的点上，可以看到与该点相连的点，其含义为与该包共同出现过的包。

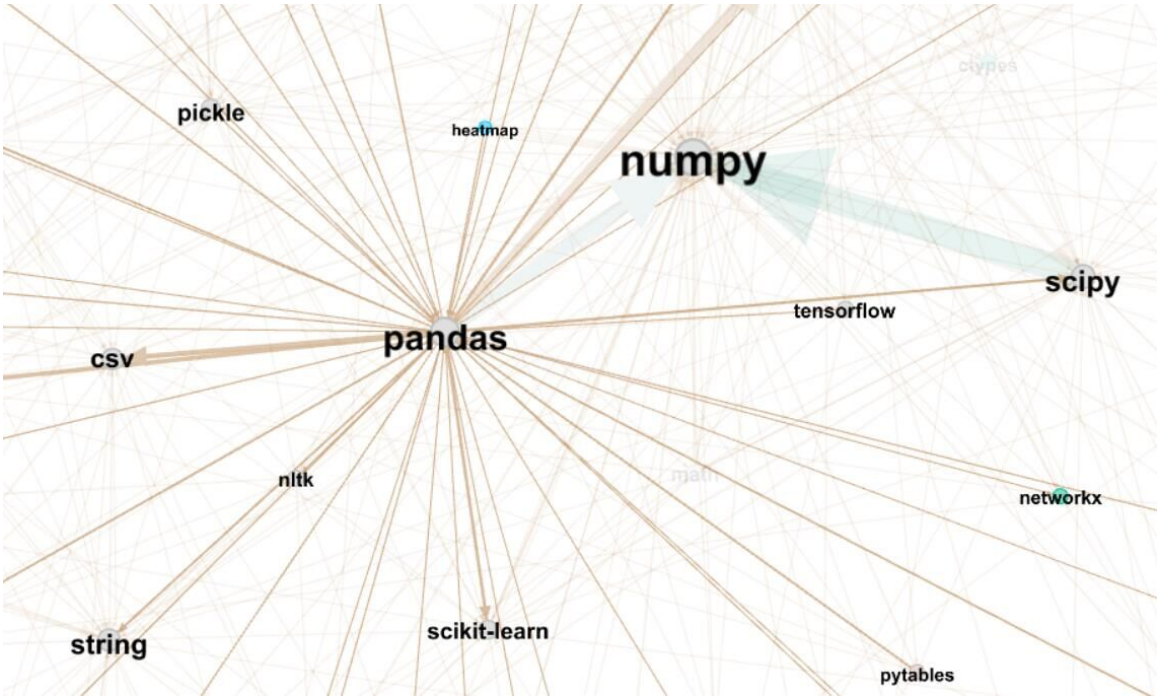

图 6: 举例：与Pandas有关的包

将GitHub上爬取的数据也分别进行可视化，结果如下：

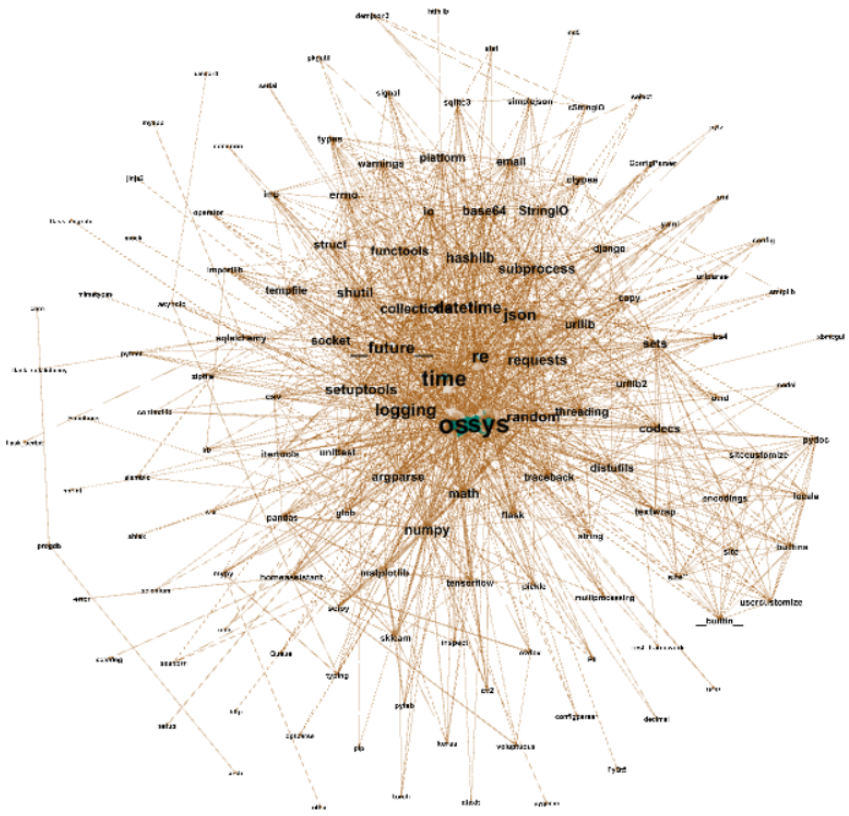

图 7: GitHub上python工程中import的包关系网络图

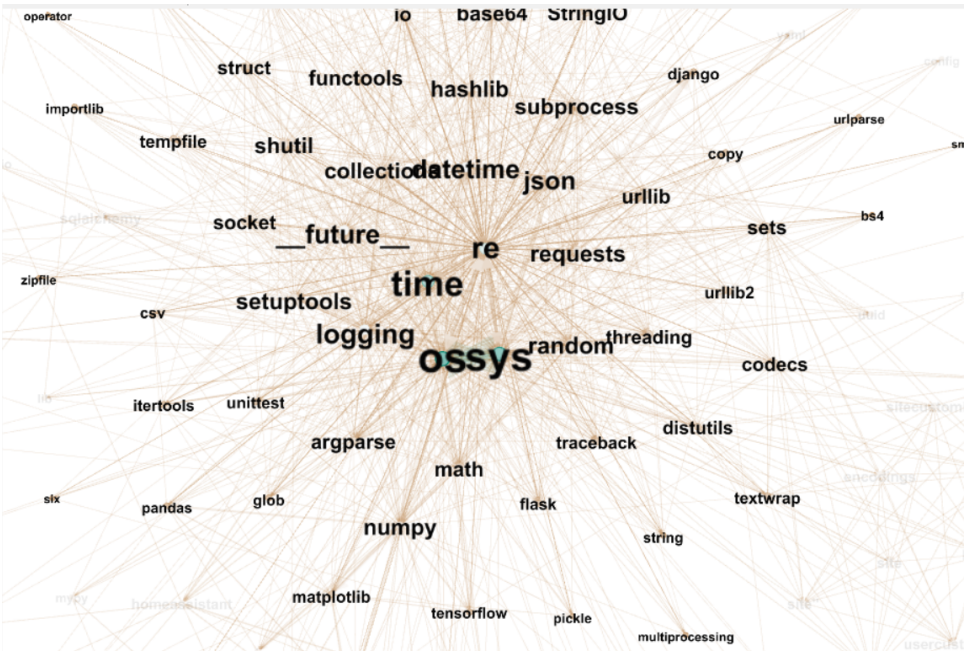

图 8: GitHub上python工程中与re有关的包

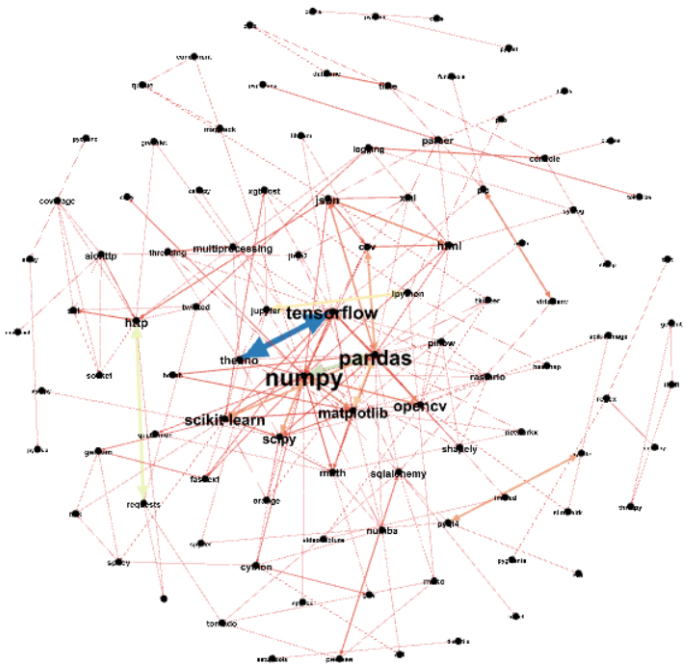

图 9: GitHub上仓库标签中提及的包的关系网络图

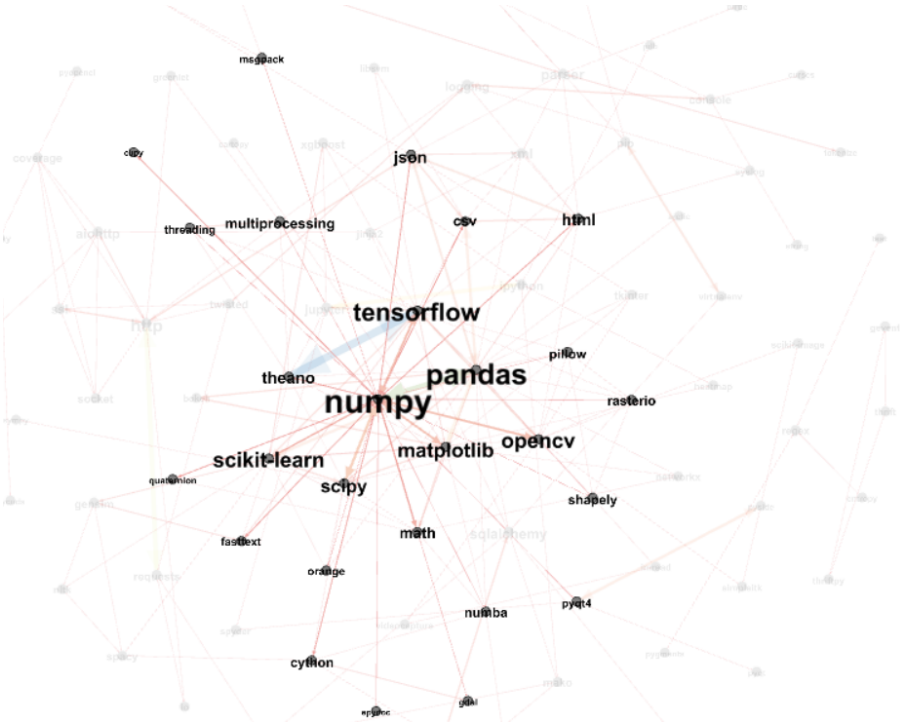

图 10: GitHub上仓库标签中与numpy有关的包

其中，最经常出现在一起的结果分别如下：

表 6: 统计StackOverflow中同一问题下包出现情况

序号	package1	package2	出现次数
1	numpy	scipy	790
2	numpy	pandas	430
3	matplotlib	numpy	384
4	matplotlib	pandas	288
5	matplotlib	scipy	146
6	pyqt4	pyqt	146
7	pip	virtualenv	138
8	csv	pandas	126
9	datetime	pandas	104
10	regex	string	92
11	cython	numpy	92
12	scikit-learn	numpy	90
13	opencv	numpy	86
14	setuptools	pip	64
15	matplotlib	ipython	62
16	pyqt	pyside	62
17	setuptools	distutils	60
18	scikit-learn	pandas	60
19	numpy	math	60
20	pandas	ipython	56
21	jupyter	ipython	54
22	csv	numpy	54
23	datetime	time	50
24	pandas	scipy	50
25	tkinter	matplotlib	44
26	pip	numpy	44

表 7: 同时出现在一个工程中的packages统计

序号	package-1	package-2	同时出现次数
1	os	sys	283
2	os	time	118
3	os	re	101
4	time	sys	97
5	os	django	93
6	django	sys	91
7	numpy	matplotlib	86
8	re	sys	86
9	os	numpy	82
10	logging	os	73
11	os	datetime	70
12	os	json	70
13	os	setuptools	66
14	numpy	sys	66
15	datetime	sys	60
16	os	argparse	60
17	json	sys	57
18	os	__future__	57
19	time	re	55
20	os	subprocess	54
21	numpy	sklearn	53
22	time	datetime	52
23	time	json	51
24	pandas	numpy	50
25	logging	sys	49
26	os	random	49

表 8: 组合出现在标签中的packages统计

序号	package-1	package-2	同时出现次数
1	theano	tensorflow	14
2	numpy	pandas	9
3	http	requests	8
4	jupyter	ipython	7
5	pandas	matplotlib	6
6	numpy	scipy	5
7	pyside	pyqt4	5
8	pandas	scikit-learn	5
9	scipy	numpy	5
10	numpy	opencv	4
11	numpy	tensorflow	4
12	json	html	4
13	opencv	tensorflow	4
14	pip	virtualenv	4
15	tensorflow	numpy	4
16	numpy	scikit-learn	3
17	csv json	3	
18	threading	multiprocessing	3

其中，在GitHub的工程中，与系统相关的包（如，os,time,sys）经常出现；与科学计算，数据处理相关的包（如，numpy, matplotlib, pandas）也经常一起出现。

在StackOverflow的提问里，与科学计算，数据处理相关的包（如，numpy, matplotlib, pandas, scipy）也经常一起出现在同一个问题中。

5.3 其他结果

tags中最多被提到的话题 统计GitHub仓库所有标签的出现次数，结果如下：

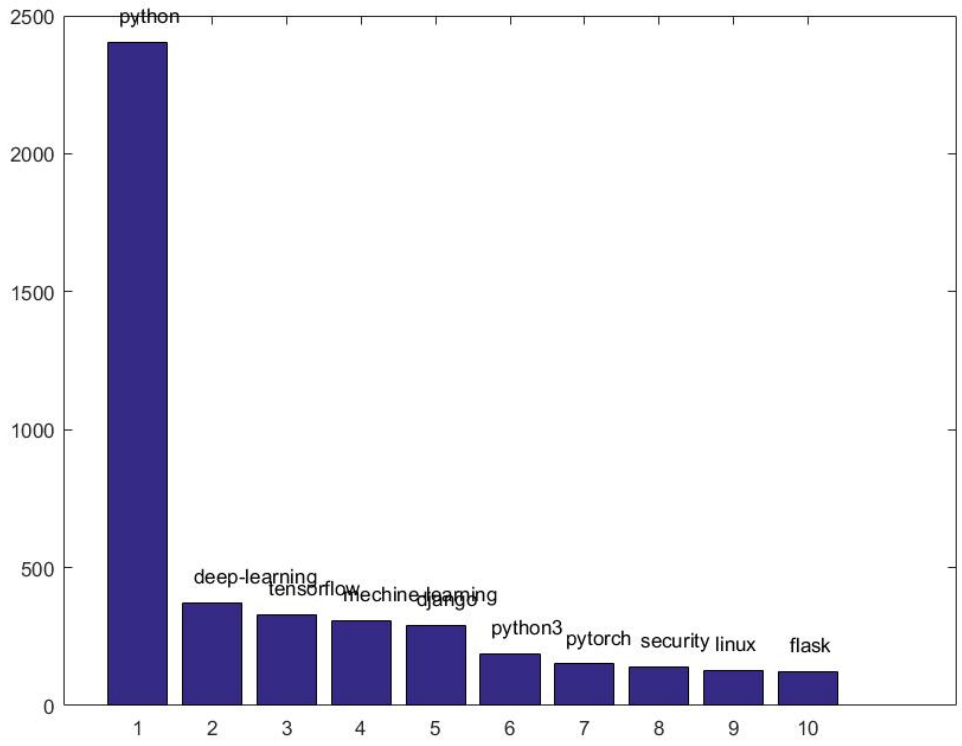

图 11: GitHub仓库所有标签的出现次数直方图

表 9: GitHub仓库所有标签的出现次数

序号	标签	出现次数
1	python	2403
2	deep-learning	373
3	tensorflow	330
4	machine-learning	307
5	django	289
6	python3	188
7	pytorch	151
8	security	139
9	linux	128
10	flask	124
11	cli	99
12	keras	97
13	docker	93
14	nlp	90
15	data-science	82
16	natural-language-processing	77
17	computer-vision	76
18	python-3	72
19	asyncio	71
20	neural-network	69

以上结果中，最常被使用的标签如下。其中，深度学习相关标签（如，deep-learning, pytorch, tensorflow, neural-network, keras, gan等），占了很大一部分；网页制作框架（如，django, flask），也出现了较多次。

被使用最多的许可证 统计GitHub仓库中被使用许可证的频率，结果如下： 详细结果

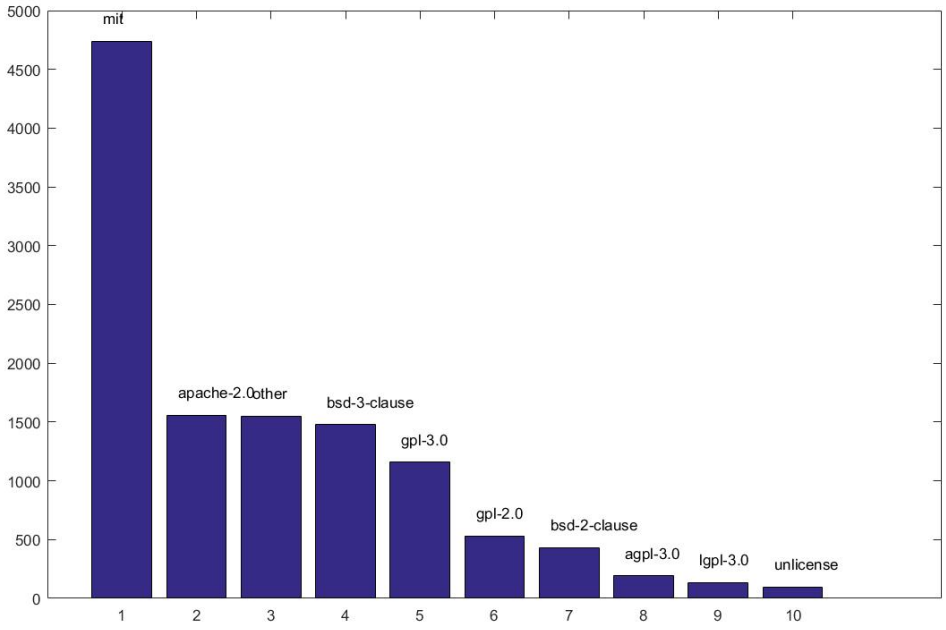

图 12: GitHub仓库中使用许可证的统计直方图

如下：

其中mit许可证占了超过三分之一的比例，被使用最多。

表 10: GitHub仓库中使用许可证的统计

许可证名	仓库数	百分比
MIT License	4736	38.90%
Apache License 2.0	1558	12.80%
Other	1551	12.74%
BSD 3-clause ″New″ or ″Revised″ License	1478	12.14%
GNU General Public License v3.0	1157	9.50%
GNU General Public License v2.0	532	4.37%
BSD 2-clause ″Simplified″ License	433	3.56%
GNU Affero General Public License v3.0	191	1.57%
GNU Lesser General Public License v3.0	133	1.09%
The Unlicense	96	0.79%
ISC License	80	0.66%
GNU Lesser General Public License v2.1	67	0.55%

6 总结

通过这门课，我们体会到了运用自己学到的知识（和未学到的、需要根据自己的需要去学习的）写一个小程序、分析出一定的结果是一种怎样的感觉。数据中蕴含着无穷的信息，而如何获取这些信息，则是一门令人惊叹、蕴含无穷乐趣的科学。在这次的作业中，我们首先便确定了要分析开源软件仓库，因为Python的一大特色便是它丰富灵活的第三方库。

我们的作业有着不少特色：

- 我们选取的数据来源，Github和Stackoverflow都提供了全面、典型的Python包数据，为我们结论的可信性奠定了基础；
- 我们直接从Github的工程文件中提取了import信息，相较于普通的爬虫，这部分数据可能更精确；
- 我们采取的可视化技术较为先进精美。

但完成作业的过程并不顺利，我们的程序也存在许多缺点：

- 我们认为Github上的工程是研究这一题目的最好材料，但逐一爬取工程文件、获取其import数据的方式效率确实有些低，我们会继续研究高效的方法。
- 时间所迫，我们并没有使用全部的信息，数据仍有较强可分析性。