정보교육을 위한 파이썬

정보 탐색

Version 0.0.9-d2

저자: Charles Severance
번역: 이광춘, 한정수
(xwmooc)

Printing history:

October 2013: Major revision to Chapters 13 and 14 to switch to JSON and use OAuth. Added new chapter on Visualization.

September 2013: Published book on Amazon CreateSpace

January 2010: Published book using the University of Michigan Espresso Book machine.

December 2009: Major revision to chapters 2-10 from *Think Python: How to Think Like a Computer Scientist* and writing chapters 1 and 11-15 to produce *Python for Informatics: Exploring Information*

June 2008: Major revision, changed title to *Think Python: How to Think Like a Computer Scientist*.

August 2007: Major revision, changed title to *How to Think Like a (Python) Programmer*.

April 2002: First edition of *How to Think Like a Computer Scientist*.

Chapter 1

데이터 시각화

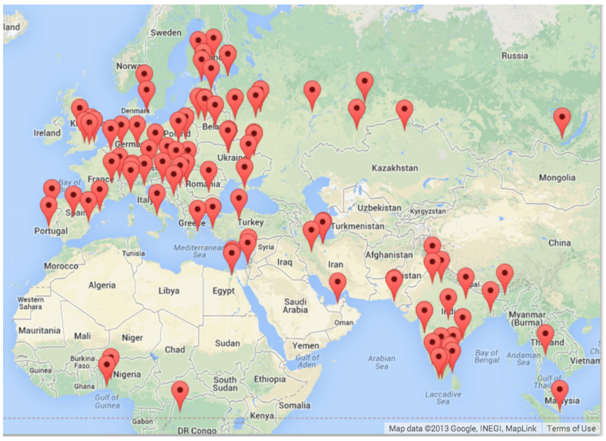
지금까지 파이썬 언어를 학습했고, 데이터를 다루기 위해서 파이썬, 네트워크, 그리고 데이터베이스를 어떻게 사용하는도 배웠다.

이번장에서는 데이터를 관리하고 시각화하기 위해서 이들 모두를 합친 완전한 세개의 응용프로그램을 살펴볼 것이다. 실제 문제를 해결하는데 시작할 수 있는 샘플 코드로 응용프로그램을 사용할 수도 있다.

각 응용프로그램은 ZIP 파일로 압축되어서 다운로드 받아 여러분의 컴퓨터에 압축을 풀고 실행한다.

1.1 지리정보 데이터로 구글맵 생성하기

이번 프로젝트에서 구글 지오코딩 API를 사용해서 사용자가 입력한 대학교 이름의 지리 정보를 정리하고나서 구글 지도에 데이터를 표시한다.



시작하기 위해서 다음 url에서 응용프로그램을 다운로드한다.

```
www.py4inf.com/code/geodata.zip
```

해결할 첫번째 문제는 무료 구글 지오코딩 API가 하루에 요청횟수에 제한이 있다는 것이다. 그래서, 만약 많은 데이터가 있다면, 여러번 중지하고 다시 시작하는 프로세스가 필요하다. 그래서, 문제를 두 단계로 나누었다.

첫번째 단계에서, **where.data** 파일에 ”설문(survey)” 데이터를 받아서, 한번에 한줄씩 읽고 구글에서 지리정보를 자져오고 **geodata.sqlite** 데이터베이스에 저장한다. 각 사용자가 입력한 위치에 대한 지오코딩 API를 사용하기 전에, 특정 입력 라인에 데이터가 있는지를 알기 위해서 간단하게 확인한다. 데이터베이스는 지오 코딩 데이터의 로컬 ”캐쉬(cache)”처럼 동작해서 두번 동일한 데이터에 대해서는 구굴에 요청하지 않도록 한다.

geodata.sqlite 파일을 제거함으로써 언제라도 프로세스를 다시 시작할 수 있다.

geoload.py 프로그램을 실행한다. **where.data**에 입력 라인을 읽고, 각 라인에 대해서 데이터베이스에 이미 존재하는지를 확인한다. 만약 위치에 대한 데이터가 없다면, 지오코딩 API를 호출해서 데이터를 가져오고 데이터베이스에 저장한다.

데이터베이스에 약간의 데이터가 이미 존재한 후에 샘플로 실행한 결과가 다음에 있다.

```
Found in database  Northeastern University
Found in database  University of Hong Kong, ...
Found in database  Technion
Found in database  Viswakarma Institute, Pune, India
Found in database  UMD
Found in database  Tufts University

Resolving Monash University
Retrieving http://maps.googleapis.com/maps/api/
    geocode/json?sensor=false&address=Monash+University
Retrieved 2063 characters {    "results" : [
{u'status': u'OK', u'results': ... }

Resolving Kokshetau Institute of Economics and Management
Retrieving http://maps.googleapis.com/maps/api/
    geocode/json?sensor=false&address=Kokshetau+Inst ...
Retrieved 1749 characters {    "results" : [
{u'status': u'OK', u'results': ... }
...
```

첫 5 지점은 이미 데이터베이스에 있으므로 건너뛴다. 프로그램은 가져오지 못한 위치 정보를 찾아 스캔하고 정보를 가져온다.

geoload.py는 언제라도 멈출 수 있다. 매번 실행에 대해서 지오코딩 API 호출의 횟수를 제한하기 위한 카운터가 있다. **where.data**가 수백개의 데이터 항목만 있다면, 하루 사용량 한계에 부딪치지 않을 것이지만, 만약 더 많은 데이터가 있다면, 입력데이터에 대한 모든 지리정보 데이터를 가진 데이터베이스를 만들기 위해서 몇일에 걸쳐서 여러번 실행할지도 모른다.

데이터를 **geodata.sqlite**에 적재하면, **geodump.py** 프로그램을 사용하여 데이터를 시각화할 수 있다. 프로그램이 데이터베이스를 일고 위치, 위도, 경도를 실행가능한 자바스크립 코드로 **where.js**에 쓴다.

geodump.py 프로그램 실행결과는 다음과 같다.

```
Northeastern University, ... Boston, MA 02115, USA 42.3396998 -71.08975
Bradley University, 1501 ... Peoria, IL 61625, USA 40.6963857 -89.6160811
...
Technion, Viazman 87, Kesalsaba, 32000, Israel 32.7775 35.0216667
Monash University Clayton ... VIC 3800, Australia -37.9152113 145.134682
Kokshetau, Kazakhstan 53.2833333 69.3833333
...
12 records written to where.js
Open where.html to view the data in a browser
```

where.html 파일은 HTML과 자바스크립트로 구성되어서 구글 맵을 시각화한다. **where.js**에 가장 최신 데이터를 읽어서 시각화할 데이터로 사용한다. 다음에 **where.js** 파일 형식이 있다.

```
myData = [
[42.3396998,-71.08975, 'Northeastern Uni ... Boston, MA 02115'],
[40.6963857,-89.6160811, 'Bradley University, ... Peoria, IL 61625, USA'],
[32.7775,35.0216667, 'Technion, Viazman 87, Kesalsaba, 32000, Israel'],
   ...
];
```

리스트의 리스트를 담고 있는 자바스크립트다. 자바스크립트 리스트 상수에 대한 구분은 파이썬과 매우 유사하여, 구문이 여러분에게 매우 친숙할 것이다.

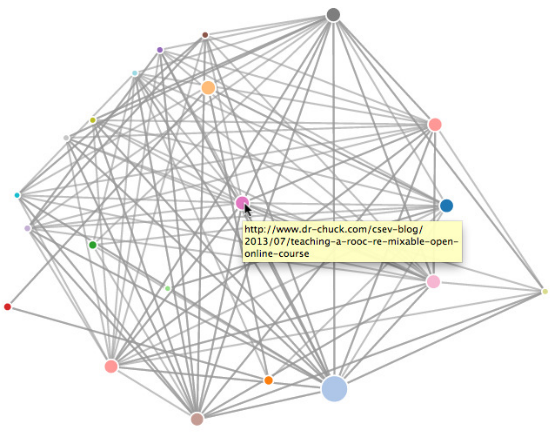
위치를 보기 위해서 브라우저에 **where.html**을 연다. 각 맵 핀을 여기저기 돌아다니면서 지오코딩 API가 사용자가 입력한 것에 대해서 반환한 위치를 찾는다. **where.html** 파일을 열었을 때 어떤 데이터도 볼 수 없다면, 자바스크립트나 브라우저의 개발자 콘솔을 확인한다.

1.2 네트워크와 상호 연결 시각화

In this application, we will perform some of the functions of a search engine. We will first spider a small subset of the web and then run a simplified version of the Google page rank algorithm to determine which pages are most highly connected and then visualize the page rank and connectivity of our small corner of the web. We will use the D3 JavaScript visualization library `http://d3js.org/` to produce the visualization output.

You can download and extract this application from:

```
www.py4inf.com/code/pagerank.zip
```


http://www.dr-chuck.com/csev-blog/
2013/07/teaching-a-rooc-re-mixable-open-
online-course

The first program (**spider.py**) program crawls a web site and pulls a series of pages into the database (**spider.sqlite**), recording the links between pages. You can restart the process at any time by removing the **spider.sqlite** file and re-running **spider.py**.

```
Enter web url or enter: http://www.dr-chuck.com/
['http://www.dr-chuck.com']
How many pages:2
1 http://www.dr-chuck.com/ 12
2 http://www.dr-chuck.com/csev-blog/ 57
How many pages:
```

In this sample run, we told it to crawl a website and retrieve two pages. If you restart the program and tell it to crawl more pages, it will not re-crawl any pages already in the database. Upon restart it goes to a random non-crawled page and starts there. So each successive run of **spider.py** is additive.

```
Enter web url or enter: http://www.dr-chuck.com/
['http://www.dr-chuck.com']
How many pages:3
3 http://www.dr-chuck.com/csev-blog 57
4 http://www.dr-chuck.com/dr-chuck/resume/speaking.htm 1
5 http://www.dr-chuck.com/dr-chuck/resume/index.htm 13
How many pages:
```

You can have multiple starting points in the same database - within the program these are called “webs”. The spider chooses randomly amongst all non-visited links across all the webs as the next page to spider.

If you want to dump the contents of the **spider.sqlite** file, you can run **spdump.py** as follows:

```
(5, None, 1.0, 3, u'http://www.dr-chuck.com/csev-blog')
(3, None, 1.0, 4, u'http://www.dr-chuck.com/dr-chuck/resume/speaking.htm')
(1, None, 1.0, 2, u'http://www.dr-chuck.com/csev-blog/')
(1, None, 1.0, 5, u'http://www.dr-chuck.com/dr-chuck/resume/index.htm')
4 rows.
```

This shows the number of incoming links, the old page rank, the new page rank, the id of the page, and the url of the page. The **spdump.py** program only shows pages that have at least one incoming link to them.

Once you have a few pages in the database, you can run page rank on the pages using the **sprank.py** program. You simply tell it how many page rank iterations to run.

```
How many iterations:2
1 0.546848992536
2 0.226714939664
[(1, 0.559), (2, 0.659), (3, 0.985), (4, 2.135), (5, 0.659)]
```

You can dump the database again to see that page rank has been updated:

```
(5, 1.0, 0.985, 3, u'http://www.dr-chuck.com/csev-blog')
(3, 1.0, 2.135, 4, u'http://www.dr-chuck.com/dr-chuck/resume/speaking.htm')
(1, 1.0, 0.659, 2, u'http://www.dr-chuck.com/csev-blog/')
(1, 1.0, 0.659, 5, u'http://www.dr-chuck.com/dr-chuck/resume/index.htm')
4 rows.
```

You can run **sprank.py** as many times as you like and it will simply refine the page rank each time you run it. You can even run **sprank.py** a few times and then go spider a few more pages sith **spider.py** and then run **sprank.py** to re-converge the page rank values. A search engine usually runs both the crawling and ranking programs all the time.

If you want to restart the page rank calculations without re-spidering the web pages, you can use **spreset.py** and then restart **sprank.py**.

```
How many iterations:50
1 0.546848992536
2 0.226714939664
3 0.0659516187242
4 0.0244199333
5 0.0102096489546
6 0.00610244329379
...
42 0.000109076928206
43 9.91987599002e-05
44 9.02151706798e-05
45 8.20451504471e-05
46 7.46150183837e-05
47 6.7857770908e-05
48 6.17124694224e-05
49 5.61236959327e-05
50 5.10410499467e-05
[(512, 0.0296), (1, 12.79), (2, 28.93), (3, 6.808), (4, 13.46)]
```

For each iteration of the page rank algorithm it prints the average change in page rank per page. The network initially is quite unbalanced and so the individual page rank values change wildly between iterations. But in a few short iterations, the page rank converges. You should run **prank.py** long enough that the page rank values converge.

If you want to visualize the current top pages in terms of page rank, run **spjson.py** to read the database and write the data for the most highly linked pages in JSON format to be viewed in a web browser.

```
Creating JSON output on spider.json...
How many nodes? 30
Open force.html in a browser to view the visualization
```

You can view this data by opening the file **force.html** in your web browser. This shows an automatic layout of the nodes and links. You can click and drag any node and you can also double click on a node to find the URL that is represented by the node.

If you re-run the other utilities, re-run **spjson.py** press refresh in the browser to get the new data from **spider.json**.

1.3 Visualizing mail data

Up to this point in the book, you have become quite familiar with our **mbox-short.txt** and **mbox.txt** data files. Now it is time to take our analysis of e-mail data to the next level.

In the real world, sometimes you have to pull down mail data from servers and that might take quite some time and the data might be inconsistent, error filled and need a lot of cleanup or adjustment. In this section, we work with an application that is the most complex so far and pull down nearly a gigabyte of data and visualize it.

You can download this application from:

```
www.py4inf.com/code/gmane.zip
```

We will be using data from a free e-mail list archiving service called `www.gmane.org`. This service is very popular with open-source projects because it provides a nice searchable archive of their e-mail activity. They also have a very liberal policy regarding accessing their data through their API. They have no rate limits, but ask that you don't overload their service and take only the data you need. You can read gmane's terms and conditions at this page:

`http://gmane.org/export.php`

It is very important that you make use of the gmane.org data responsibly by adding delays to your access of their services and spreading long-running jobs over a longer period of time. Do not abuse this free service and ruin it for the rest of us.

When the Sakai e-mail data was spidered using this software, it produced nearly a Gigabyte of data and took a number of runs on several days. The file **README.txt** in the above ZIP may have instructions as to how you can download a pre-spidered copy of the **content.sqlite** file for a majority of the Sakai e-mail corpus so you don't have to spider for five days just to run the programs. If you download the pre-spidered content, you should still run the spidering process to catch up with more recent messages.

The first step is to spider the gmane repository. The base URL is hard-coded in the **gmane.py** and is hard-coded to the Sakai developer list. You can spider another repository by changing that base url. Make sure to delete the **content.sqlite** file if you switch the base url.

The **gmane.py** file operates as a responsible caching spider in that it runs slowly and retrieves one mail message per second so as to avoid getting throttled by gmane. It stores all of its data in a database and can be interrupted and re-started as often as needed. It may take many hours to pull all the data down. So you may need to restart several times.

Here is a run of **gmane.py** retrieving the last five messages of the Sakai developer list:

```
How many messages:10
http://download.gmane.org/gmane.comp.cms.sakai.devel/51410/51411 9460
    nealcaidin@sakaifoundation.org 2013-04-05 re: [building ...
http://download.gmane.org/gmane.comp.cms.sakai.devel/51411/51412 3379
    samuelgutierrezjimenez@gmail.com 2013-04-06 re: [building ...
http://download.gmane.org/gmane.comp.cms.sakai.devel/51412/51413 9903
    da1@vt.edu 2013-04-05 [building sakai] melete 2.9 oracle ...
http://download.gmane.org/gmane.comp.cms.sakai.devel/51413/51414 349265
    m.shedid@elraed-it.com 2013-04-07 [building sakai] ...
http://download.gmane.org/gmane.comp.cms.sakai.devel/51414/51415 3481
    samuelgutierrezjimenez@gmail.com 2013-04-07 re: ...
http://download.gmane.org/gmane.comp.cms.sakai.devel/51415/51416 0

Does not start with From
```

The program scans **content.sqlite** from one up to the first message number not already spidered and starts spidering at that message. It continues spidering until it has spidered the desired number of messages or it reaches a page that does not appear to be a properly formatted message.

Sometimes `gmane.org` is missing a message. Perhaps administrators can delete messages or perhaps they get lost. If your spider stops, and it seems it has hit a missing message, go into the SQLite Manager and add a row with the missing id leaving all the other fields blank and restart **gmane.py**. This will unstick the spidering process and allow it to continue. These empty messages will be ignored in the next phase of the process.

One nice thing is that once you have spidered all of the messages and have them in **content.sqlite**, you can run **gmane.py** again to get new messages as they get sent to the list.

The **content.sqlite** data is pretty raw, with an inefficient data model, and not compressed. This is intentional as it allows you to look at **content.sqlite** in the SQLite Manager to debug problems with the spidering process. It would be a bad idea to run any queries against this database as they would be quite slow.

The second process is to run the program **gmodel.py**. This program reads the rough/raw data from **content.sqlite** and produces a cleaned-up and well-modeled version of the data in the file **index.sqlite**. The file index.sqlite will be much smaller (often 10X smaller) than **content.sqlite** because it also compresses the header and body text.

Each time **gmodel.py** runs - it deletes and re-builds **index.sqlite**, allowing you to adjust its parameters and edit the mapping tables in **content.sqlite** to tweak the data cleaning process. This is a sample run of **gmodel.py**. It prints a line out each time 250 mail messages are processed so you can see some progress happening as this program may run for a while processing nearly a Gigabyte of mail data.

```
Loaded allsenders 1588 and mapping 28 dns mapping 1
1 2005-12-08T23:34:30-06:00 ggolden22@mac.com
251 2005-12-22T10:03:20-08:00 tpamsler@ucdavis.edu
501 2006-01-12T11:17:34-05:00 lance@indiana.edu
751 2006-01-24T11:13:28-08:00 vrajgopalan@ucmerced.edu
...
```

The **gmodel.py** program handles a number of data cleaning tasks.

Domain names are truncated to two levels for .com, .org, .edu, and .net. Other domain names are truncated to three levels. So si.umich.edu becomes umich.edu and caret.cam.ac.uk becomes cam.ac.uk. Also e-mail addresses are forced to lower case and some of the @gmane.org address like the following

```
   arwhyte-63aXycvo3TyHXe+LvDLADg@public.gmane.org
```

are converted to the real address whenever there is a matching real e-mail address elsewhere in the message corpus.

If you look in the **content.sqlite** database there are two tables that allow you to map both domain names and individual e-mail addresses that change over the lifetime of the e-mail list. For example, Steve Githens used the following e-mail addresses as he changed jobs over the life of the Sakai developer list:

```
s-githens@northwestern.edu
sgithens@cam.ac.uk
swgithen@mtu.edu
```

We can add two entries to the Mapping table in **content.sqlite** so **gmodel.py** will map all three to one address:

```
s-githens@northwestern.edu ->  swgithen@mtu.edu
sgithens@cam.ac.uk -> swgithen@mtu.edu
```

You can also make similar entries in the DNSMapping table if there are multiple DNS names you want mapped to a single DNS. The following mapping was added to the Sakai data:

```
iupui.edu -> indiana.edu
```

So all the accounts from the various Indiana University campuses are tracked together.

You can re-run the **gmodel.py** over and over as you look at the data, and add mappings to make the data cleaner and cleaner. When you are done, you will have a nicely indexed version of the e-mail in **index.sqlite**. This is the file to use to do data analysis. With this file, data analysis will be really quick.

The first, simplest data analysis is to determine ”who sent the most mail?” and ”which organization sent the most mail”? This is done using **gbasic.py**:

```
How many to dump? 5
Loaded messages= 51330 subjects= 25033 senders= 1584

Top 5 Email list participants
steve.swinsburg@gmail.com 2657
azeckoski@unicon.net 1742
ieb@tfd.co.uk 1591
csev@umich.edu 1304
david.horwitz@uct.ac.za 1184

Top 5 Email list organizations
gmail.com 7339
umich.edu 6243
uct.ac.za 2451
indiana.edu 2258
unicon.net 2055
```

Note how much more quickly **gbasic.py** runs compared to **gmane.py** or even **gmodel.py**. They are all working on the same data, but **gbasic.py** is using the compressed and normalized data in **index.sqlite**. If you have a lot of data to manage, a multi-step process like the one in this application may take a little longer to develop, but will save you a lot of time when you really start to explore and visualize your data.

You can produce a simple visualization of the word frequency in the subject lines in the file **gword.py**:

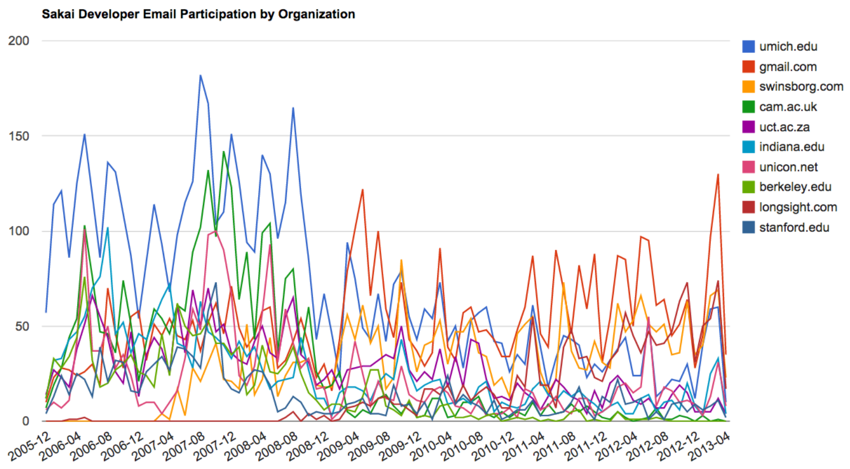
```
Range of counts: 33229 129
Output written to gword.js
```

This produces the file **gword.js** which you can visualize using **gword.htm** to produce a word cloud similar to the one at the beginning of this section.

A second visualization is produced by **gline.py**. It computes e-mail participation by organizations over time.

```
Loaded messages= 51330 subjects= 25033 senders= 1584
Top 10 Oranizations
['gmail.com', 'umich.edu', 'uct.ac.za', 'indiana.edu',
'unicon.net', 'tfd.co.uk', 'berkeley.edu', 'longsight.com',
'stanford.edu', 'ox.ac.uk']
Output written to gline.js
```

Its output is written to **gline.js** which is visualized using **gline.htm**.



This is a relatively complex and sophisticated application and has features to do some real data retrieval, cleaning and visualization.