暨南大学本科实验报告专用纸

课程名称 数值分析 成绩评定

实验项目名称 Google PageRank 指导教师 方良达

实验项目编号 04 实验项目类型 验证型

实验地点 N515 学生姓名 黄芝琪,李文灏 学号 2015052289,2015052377

学院 信息科学技术 系 计算机科学 专业 计算机科学技术专业

实验时间 2017 年 12 月 15 日 上 午

Ⅰ、Problem

1. Implement a simple web crawler;
2. Acquire the google matrix of the first 1000 web pages from http://www.scutde.net via the above web crawler, and give their adjacency matrix
3. Implement the Power Method
4. Compute the dominant eigenvector of the google matrix
5. List the top 20 web pages.

Ⅱ、The basic idea of the algorithm

Google's PAGERANK uses the hyperlink structure of the Web to view in-links into a page as a recommendation of that page from the author of the in-linking page and In-links. Hence,*Good* pages will have in-links from *good* pages;and conversely, *good* pages will have out-links to *good* pages. And in-links from *good* pages should carry more weight than in-links from *marginal* pages.

Google PAGERANK have following characters:

1. Pages vote for the importance of other pages by linking to them.
The more in-links a page has, the more important it is!
2. One person, one vote.
If a page has more than one out-link, it must split its vote!
3. A link to page *k* from an important page should boost page *k*'s importance score more than a link from an unimportant page.
It matters who your friends and supporters are!

Consider the web surfer can choose to stay put or jump to any other node with *equal probability*, We should modify adjacency matrix with equation r=(aB+(1-a)E)r=Gr,where a=0.85 and E=*e*eT/*m.

Ⅲ、Experimental procedures

Step 1: find the first 1000 web pages from http://www.scutde.net via the above crawler, store them with a array web Link[ ];

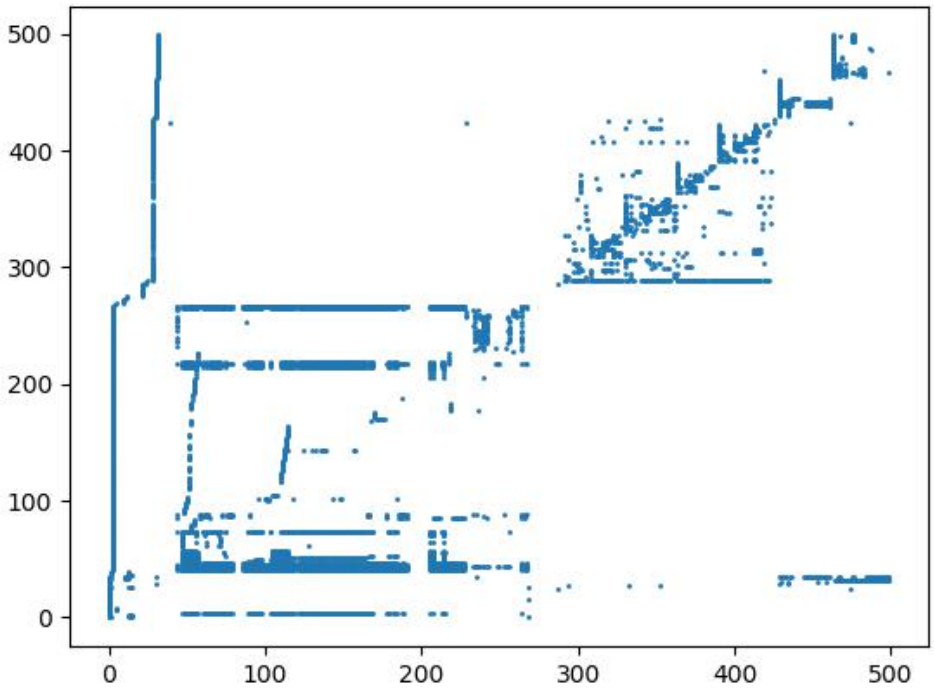
Step 2: traverse web Link[ ] to computing the 1000 web Pages' connectivity, store them with a 1000*1000 2-dimension array aMatrix[ ][ ], which is adjacency matrix;

Step 3: recording and weighting the "votes": for every page, use (aB+(1-a)E) to change aMartix[ ][ ] to gMatrix[ ][ ], where B=aMatrix,E=*e*eT/*m and a=0.85;

Step 4: give a initial x[ ], use Power Iteration to compute dominant eigenvector of gMatrix[ ][ ], compute for 5000 times, and finally x[ ] is dominant eigenvector of gMatrix[ ];

Step 5: sort V_dic[ ] and list the top 20 web pages.

Ⅳ、Result analysis



Adjacency matrix for the starting page http://www.scutde.net

```
1 : http://syss.12371.cn/
2 : http://www.scutde.net
3 : http://www.5184.com/
4 : http://syss.12371.cn/gdcz/
5 : http://www.chsi.com.cn/
6 : http://liuxue.scutde.net/
7 : http://liuxue.scutde.net/modules/pages/enrollment/gjxm.aspx
8 : http://liuxue.scutde.net/modules/pages/enrollment/dx.aspx
9 : http://www.scutde.net/zs/ggtz/wljy/1buc1av2t3n16.xhtml
10 : http://www.cdce.cn/
11 : http://liuxue.scutde.net
12 : http://www.chsi.cn/
13 : http://www.scutde.net/jxcg/index.htm
14 : http://www.5184.com/zikao/skb/201712/43997.html
15 : http://www.5184.com/zikao/bidu/201705/42808.html
16 : http://www.scutde.net/jxcg/
17 : https://daiban.5184.com/
18 : http://www.cdce.cn
19 : http://syss.12371.cn/gdcz/zgjrqy/
20 : http://syss.12371.cn/gdcz/zggx/
```

Top 20 pages that have the most weight

Ⅵ、Experimental summary

This experiment did take me plenty of energy. At the beginning, I have to write a spider program to spider 500 websites using breadth first search starting out with the index page of Shanghai Jiao Tong University. Then, I use Python instead of Matlab because of the convenience of it, which can be reflect in the method dealing with network. In a word, this experiment benefits me quite a lot.

Ⅶ、Appendix: Source code

Spider

```
import requests
import re

'''
init web pages query
'''
websites = []
tmp_websites = []

def find_all_url(content):
    '''
```

```
    find all urls in the content and return a list of them
    '''
    pattern_url                                                    =
    re.compile(r'(?<=href=\").+?(?=\")|(?<=href=\').+?(?=\')')
    result1 = pattern_url.findall(content)
    #print result1
    for x in result1:
      tmp_websites.append(x)

def find_url(url):
    if url in websites:
      return
    try:
      r = requests.get(url,timeout=5)
    except:
      print 'timout'
      return
    else:
      print 'good'
      text = r.content
      #regex ,to find out all valid url,add to tmp_query
      tmp_url_list = find_all_url(text)
      #print text
      #tmp url added in websites query
      websites.append(url)



if __name__ =='__main__':
    init_url = 'http://www.sjtu.edu.cn/'
    tmp_websites.append(init_url)
    #websites.append(init_url)
    print tmp_websites[0]
    #tmp_url = tmp_websites[0]



    while len(websites)<500 and len(tmp_websites)>=1:
      print("spidering the {}/500 page ".format(len(websites)+1))
      #get the first url of websites query
      tmp_url = tmp_websites[0]
      #delete the first item
      del tmp_websites[0]
      find_url(tmp_url)
```

```
        #print websites


      print 'saving data***********'
      file                                                              =
      open('result_'+init_url.replace(init_url,'shjd')+'.txt','w+')
      for x in websites:
        file.write(x + '\n')
      file.close()
```

Plot and ranking preparation

```
import matplotlib.pyplot as plt
import numpy as np
import requests
import re

f = open('result__www.scutde.net.txt','r')

urls = []

while True:
      url = f.readline().rstrip('\n')

      if url:
        urls.append(url)
      else:
        break

relation_matrix = np.zeros((len(urls),len(urls)))
#generate a matrix to be plotted
for i in range(0,len(urls)):
      try:
        r = requests.get(urls[i],timeout=5)
      except :
        print 'timeout'
        continue
      # find out all urls in the specific website
      print("good,dealing the {}/500 url".format(i))
      content = r.content
      pattern_url                                                       =
      re.compile(r'(?<=href=\").+?(?=\")|(?<=href=\').+?(?=\')')
      result1 = pattern_url.findall(content)
      #if each item in urls is included by the specific website  content
```

```
     urls
     for j in range(0,len(urls)):
       if urls[j] in result1:
           relation_matrix[i][j] = 1

#save the matrix
np.savetxt('result_matrix.txt',relation_matrix)
```

Plot and ranking process

```
# -*- coding:utf-8 -*-

import numpy as np
import matplotlib.pylab as plt
import operator

#load the relation_matrix
relation_matrix = np.loadtxt("result_matrix.txt")

#init the x[] and y[]
x = list(np.zeros((1)))
y = list(np.zeros((1)))

for i in range(0,500):
    for j in range(0,500):
        if relation_matrix[i][j] == 1:
            x.append(i)
            y.append(j)


# plot the x[],y[]
plt.scatter(x,y,s = 1)
plt.show()

#init the matrix L
L = np.zeros((500,500))
for i in range(0,len(x)):
    L[int(x[i])][int(y[i])] = 1

#translate L into A
for i in range(0,500):
    sum = 0
    for j in L[:,i]:
        sum += j
```

```
    if not  sum == 0:
        L[:,i] = L[:,i]/sum
A = L

A = np.array(A)
#power iteration for G
X = np.zeros((500,1))
#init a random vector
X[0][0] = 1
# init e and k
e = np.ones((500,1))
k = 0.85

for i in range(0,5000):
    Y = np.dot(A,X)

    B = 1 - k*np.sum(Y)
    X = k*Y + (B/500)*e
# V is the result
V = X

#load the urls from file
f = open('result__www.scutde.net.txt','r')
urls = []
while True:
    url = f.readline().rstrip('\n')
    if url:
        urls.append(url)
    else:
        break

#generate a dictionary that has a map from url to vector V
V_dic = {}
V = list(V)
for i in range(0,500):
    V_dic[urls[i]] = float(V[i])

#sort the V_dic by its values
sorted_result=sorted(V_dic.items(),key=operator.itemgetter(1),reverse
=True)
#output the top 20 results
for i in range(0,20):
    print('{} : {}'.format(i+1,sorted_result[i][0]))
```