MAT167 Final Project

Alexander Sun

June 2023

1 Introduction

Google's PageRank algorithm models surfing the Web as a random walk. The basic idea is to represent this random walk as a Markov Chain. The PageRank of each website is its probability in the stationary distribution. A website's PageRank can be interpreted as a measure of its popularity or importance. The PageRank algorithm can be applied to any directed graph, not just to websites.

In this project, we used PageRank to analyze the LastFM Asia Social Network, where the vertices in the directed graph are users, and the edges are follower relationships between them. We calculated the PageRank by solving a sparse linear system. We found that a small amount of users have much higher PageRank than the typical user.

2 Background

Suppose we have a set of n websites that we want to PageRank, and for each website we know what hyperlinks it contains. We represent this information with an n-by-n connectivity matrix G of a portion of the Web. So $g_{ij} = 1$ if there is a hyperlink from page j to page i, and $g_{ij} = 0$ otherwise.

The transition matrix A of the Markov Chain can be calculated by

$$a_{ij}{}^{1} = \begin{cases} pg_{ij}/c_{ij} + (1-p)/n & c_j \neq 0 \\ 1/n & c_j = 0 \end{cases}$$

where p is the probability that the random walk follows a link, and c_j is the number of hyperlinks that website j has. Note that we should choose $p < 1$, which guarantees that the *Perron–Frobenius theorem*[2] applies.

To find the stationary distribution of a, we simply solve $x = Ax$. The way we will do this in MatLab is to use $A = pGD + ez^T$, where D is the diagonal matrix

$$d_{jj} = \begin{cases} 1/c_j & c_j \neq 0 \\ 0 & c_j = 0 \end{cases}$$

[1] a_{ij} is the transition probability from j to i. This makes the linear algebra easier

[2] x = Ax exists and is unique to within a scaling factor, so there is a unique stationary distribution

e is the n-vector of all ones, and z is the vector with components

$$z_j = \begin{cases} (1-p)/n & c_j \neq 0 \\ 1/n & c_j = 0 \end{cases}$$

The equation $x = Ax$ can be rewritten as $(I - pGD)x = \gamma e$ where $\gamma = z^T x$. Since γ depends on x, and x can be scaled arbitrarily, we can set $\gamma = 1$. Thus, the problem reduces to solving $(I - pGD)x = e$. Lastly, we rescale x such that its entries sum to 1.

3 Solutions

Ex. 2.25

a) The connectivity matrix G is:

$$G = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

b) Following the procedure outlined in the Background, we find that

$$x = \begin{bmatrix} 0.1981 \\ 0.1092 \\ 0.1556 \\ 0.2037 \\ 0.1667 \\ 0.1667 \end{bmatrix}$$

c) The parameter p represents how important the connectivity matrix G is. If $p = 0$, then we do not care about G at all. But in the limit $p \to 1$, the Markov Chain approaches the random walk of a web surfer who only follows hyperlinks. The computation of PageRanks using the power method may take longer to converge, or tend to oscillate more.

Ex. 2.26

a) pagerank1.m

```
function x = pagerank1(G)
% solves the sparse linear system

% c = out-degree
[n,n] = size(G);
```

```
c = sum(G,1);

% Scale column sums to be 1 (or 0 where there are
   no out links).
k = find(c~=0);
D = sparse(k,k,1./c(k),n,n);

% Solve (I - p*G*D)*x = e
e = ones(n,1);
I = speye(n,n);
p = .85;
x = (I - p*G*D)\e;

% Normalize so that sum(x) == 1.
x = x/sum(x);
```

b) pagerank2.m

```
function x = pagerank2(G)
% uses inverse iteration

% c = out-degree
[n,n] = size(G);
c = sum(G,1);

% Scale column sums to be 1 (or 0 where there are
   no out links).
k = find(c~=0);
D = sparse(k,k,1./c(k),n,n);

% calculate z
p = .85;
z = c;
for i = 1:n
    if z == 0
        z = 1/n;
    else
        z = (1-p)/n;
    end
end

% calculate e and I
e = ones(n,1);
I = speye(n,n);

% calculate A
```

```
A = p*G*D + e*transpose(z);

% inverse iteration step
x = (I - A)\e;

% Normalize so that sum(x) == 1.
x = x/sum(x);
```

c) pagerank3.m

```
function x = pagerank3(G)
% uses the Power method

% r = in-degree
[n,n] = size(G);
c = sum(G,1);

% Find G
k = find(c~=0);
D = sparse(k,k,1./c(k),n,n);
p = .85;
G = p*G*D;

% find e and z
z = ((1-p)*(c~=0) + (c==0))/n;
e = ones(n,1);

% x is new, x2 is old
x = ones(n,1)/n;
x2 = zeros(n,1);
while max(abs(x-x2)) > .0001
    x2 = x;
    x = G*x2 + e*(z*x2);
end
```

d) pagerank1(G):

$$x = \begin{bmatrix} 0.3210 \\ 0.1705 \\ 0.1066 \\ 0.1368 \\ 0.0643 \\ 0.2007 \end{bmatrix}$$

pagerank2(G):

$$x = \begin{bmatrix} 0.3210 \\ 0.1705 \\ 0.1066 \\ 0.1368 \\ 0.0643 \\ 0.2007 \end{bmatrix}$$

pagerank3(G):

$$x = \begin{bmatrix} 0.3210 \\ 0.1706 \\ 0.1066 \\ 0.1368 \\ 0.0643 \\ 0.2008 \end{bmatrix}$$

Part 3

The dataset I picked is LastFM Asia Social Network. Here is the code I wrote in Python to parse the data:

```python
# importing module
from pandas import *
import scipy.io

# reading CSV file
data = read_csv("lastfm_asia_edges.csv")

# converting column data to list
i = data['node_2'].tolist()
i = [x+1 for x in i] # MatLab indexes from 1
j = data['node_1'].tolist()
j = [x+1 for x in j]

# save to matlab
scipy.io.savemat('i.mat', mdict={'i': i})
scipy.io.savemat('j.mat', mdict={'j': j})
```

I loaded the files i.mat and j.mat in MatLab, which I then used to get the connectivity matrix G. From there, I ran pagerank1(G). Lastly, I plotted the results. Here is all the code in MatLab:

```
>> load('i.mat')
>> load('j.mat')
>> n = 7624
>> G = sparse(i,j,1,n,n)
>> x = pagerank1(G)

>> shg
>> [sortedx, sortOrder] = sort(x, 'descend');
>> bar(sortedx)
```

```
>> title('Page Rank')
```

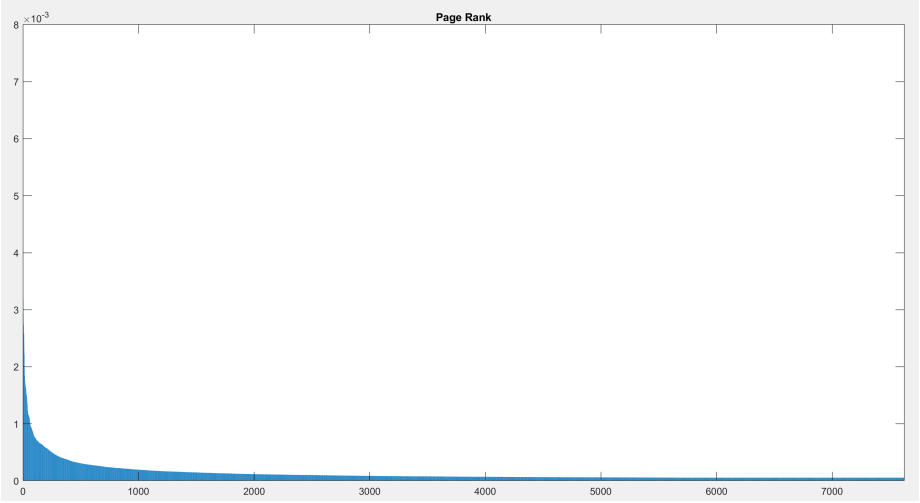

Figure 1: PageRanks of users in descending order

We see that a small amount of users have disproportionately large PageRank, which shows that a small amount of users are many times more popular than the typical user. This could reflect a few celebrities who use LastFM having a lot of followers.