ISC 5315
Applied Computational Science
Lab: Linear Algebra: Google PageRank
Due: Sep. 29, 2023

1 Introduction

Google PageRank (named after Google co-founder Larry Page) is an algorithm used to figure out the relative "importance" of a webpage by performing link analysis. Before this, most search engines used keyword density to determine the order in which to display search results. Marziah Karch presents a good non-mathematical description of the guiding principle behind PageRank (at about.com):

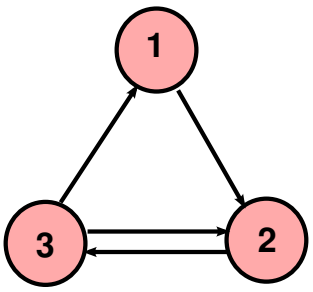
> Page and Brin's theory is that the most important pages on the Internet are the pages with the most links leading to them. PageRank thinks of links as votes, where a page linking to another page is casting a vote.
>
> This makes sense, because people do tend to link to relevant content, and pages with more links to them are usually better resources than pages that nobody links.
>
> PageRank doesn't stop there. It also looks at the importance of the page that contains the link. Pages with higher PageRank have more weight in "voting" with their links than pages with lower PageRank. It also looks at the number of links on the page casting the "vote." Pages with more links have less weight.
>
> This also makes a certain amount of sense. Pages that are important are probably better authorities in leading web surfers to better sources, and pages that have more links are likely to be less discriminating on where they're linking.

2 Goal

In this laboratory, we will explore a simple form of the PageRank model which seeks to rank webpages based on in-links and out-links, and evaluate the computational performance of two different methods for computing the PageRank. One of the methods involves solving a dense linear system, $\mathbf{Ax} = \mathbf{b}$, and the other involves finding the eigenvector corresponding to the largest eigenvalue.

3 Model

You can read the general description of the PageRank algorithm on the wikipedia entry referenced at the end of this lab. Here, I will try to describe the PageRank model with a particular simple example.



Let us assume that there are only three web-pages, numbered 1 through 3, with links between them as shown in the figure. Page 1 links to page or site 2, which in turn links to page 3. Page 3 links to both pages 1 and 3. We can write a adjacency matrix $\mathbf{A}$, which describes the structure of links.

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

A "1" in the (i, j) position indicates a link from site j to site i. Thus, the first row of the matrix tells us that only site 3 links into site 1; the first column tells us that site 1 links out only to site 2. Note that the matrix is not symmetric and that we don't consider self-links (the diagonal is zero). The number of out-links (column sums) from sites 1, 2, and 3, are $l_1 = 1$, $l_2 = 1$, and $l_3 = 2$, respectively.

In the simplest form, the PageRank of page i is given by

$$p_i = \sum_{j \to i} \frac{p_j}{l_j} \tag{1}$$

where the summation runs over all sites j that link into i. In other words, each webpage "distributes" its PageRank to its outlinks. In this particular example, this leads to the following set of equations:

$$\begin{aligned} p_1 &= \frac{p_3}{l_3} = \frac{p_3}{2} \\ p_2 &= \frac{p_1}{1} + \frac{p_3}{2} \\ p_3 &= \frac{p_2}{1} \end{aligned}$$

In order to account for an imaginary surfer who eventually stops clicking, eqn. 1 is modified by adding a damping factor d, usually set to around 0.85. Thus,

$$p_i = \frac{1-d}{n} + d \sum_{j \to i} \frac{p_j}{l_j} \tag{2}$$

where n is the total number of webpages (=3 here). Since n can be large, we would like to write this equation in matrix form.

Let $\mathbf{R}$ be a column vector of PageRanks;

$$\mathbf{R} = \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_n \end{bmatrix}$$

Further, let us modify the adjacency matrix by dividing column i by the number of outlinks l_i. In this modified adjacency matrix $\mathbf{M}$, all the columns add up to 1. For this example,

$$\mathbf{M} = \begin{bmatrix} 0.0 & 0.0 & 0.5 \\ 1.0 & 0.0 & 0.5 \\ 0.0 & 1.0 & 0.0 \end{bmatrix}$$

We can now compress the series of equations in eqn. 2 to the matrix equation

$$\mathbf{R} = \frac{1-d}{n}\mathbf{1} + d\mathbf{M}\mathbf{R} \tag{3}$$

where $\mathbf{1}$ is a $n \times 1$ vector, full of 1s.

Method 1: This directly leads us to a method, which involves solving this system of equations for $\mathbf{R}$. Note that we can write the above equation as:

$$(\mathbf{I} - d\mathbf{M})\mathbf{R} = \frac{1-d}{n}\mathbf{1} \tag{4}$$

where $\mathbf{I}$ is the $n \times n$ identity matrix.

Method 2: It can be shown that if $\mathbf{R}$ is normalized so that its entries add up to 1, and none of the columns of $\mathbf{M}$ add to zero (no dead-end sites), then we can rewrite equation 3 as:

$$\begin{aligned}\mathbf{R} &= \left(d\mathbf{M} + \frac{1-d}{n}\mathbf{E}\right)\mathbf{R} \\ &= \hat{\mathbf{M}}\mathbf{R}\end{aligned}$$

where $\mathbf{E}$ is a $n \times n$ matrix full of 1s. We can now visualize this as the eigenvalue problem:

$$\hat{\mathbf{M}}\mathbf{R} = 1\mathbf{R}$$

It can be shown that the largest eigenvalue is indeed 1, and that is what we seek. Thus, we can use the Power Method to find $\mathbf{R}$.

4 Exercises

1. Write a computer program to select a random $n \times n$ adjacency matrix with 0s and 1s as its elements. Set all the diagonal elements to zero. Note that each column should have at least one non-zero entry; if it doesn't, discard the matrix and try again.

2. Find the modified adjacency matrix $\mathbf{M}$, and use method 1 to find the PageRanks.

3. Use Power Method to find the eigenvector corresponding to largest eigenvalue (=1) of the matrix $\hat{\mathbf{M}}$. Use $|\hat{\mathbf{M}}\mathbf{R} - \lambda\mathbf{R}|$ as the termination criterion where the eigenvalue is estimated as $\lambda = \mathbf{R}^T\hat{\mathbf{M}}\mathbf{R}/(\mathbf{R}^T\mathbf{R})$.

4. Repeat the three steps above for $n = 5, 10, 50, 100, 500, 1000, 2000,$ and 5000. Plot the computational cost of both methods against n on a log-log scale. Discuss what you observe in as much detail as possible. At the very least, comment on the slopes, and which method you think is better.

5. For a given n, if all we care about is the rank order of the top ten sites, what is the most aggressive termination criterion you can get away with. In other words, we do not need to calculate accurate values for these top 10 sites, but just want a correct ordering of them. Note that the top 10 sites are corresponding to the 10 largest entries in $\mathbf{R}$. The indexes of these 10 largest entries can be extracted using the numpy.argsort() function in Python. You can first use "Method 1" to obtain the top 10 sites, and then use the power method to solve for $\mathbf{R}$ with a series of termination criteria and find the largest criterion that is

able to give the top 10 sites correctly. Report the most aggressive termination criterion δ_{max} for $n = 50, 100, 500, 1000, 2000, 5000$, and make a plot of δ_{max} versus n.

6. Search-Engine Optimization has become a cottage industry. Use the web to find out some of the common techniques used, and discuss how they "game" the algorithm.

5 References

1. Wikipedia has a nice entry, from which a lot of this material was derived.
   `http://en.wikipedia.org/wiki/PageRank`

2. Marziah Karch has a good intuitive explanation of how PageRank works.
   http://google.about.com/od/searchengineoptimization/a/pagerankexplain.htm

3. Brin and Page, **1998**, "The anatomy of a large-scale hypertextual Web search engine", *Computer Networks and ISDN Systems*, 30: 107-117.