Google PageRank

Abstract

This case study serves to introduce the basic problem of linear algebra, a *system of linear equations*. We discuss how one can solve such a system in MATLAB using either direct or iterative methods.

1 Organizing the Web

By some estimates, the World Wide Web contains more than 45 billion documents. Search engines such as Google are used to help find relevant documents in this vast collection. This is commonly done by matching a given search phrase such as "Engineering Analysis" to web pages that contain this phrase. The challenge here is that there will typically be a large number of web pages containing the search phrase, with some of these being much more useful than others. The main value a search engine gives to its users is to *rank* these results so that the most relevant are at the top of the list. This is a daunting problem given the size of the web and the diversity of the possible queries.

2 PageRank

Early search engines ranked each page by using data on a page such as the number of times a search phrase appeared. The founders of Google, Sergey Brin and Lawrence Page, came up with an alternative idea to help rank web pages while they were computer science students at Stanford. The key behind their idea, which they called PageRank, is that an important web page on a topic would have many other web pages linking to it, i.e., they use the structure of the web to rank pages.[1] In addition to the number of links, PageRank also takes into account the importance of the linking pages. The resulting ranking is one of many factors Google uses to determine the most relevant matches for a particular query. In the following we will describe a simplified version of this PageRank algorithm.

As a concrete example, consider a collection of 5 web pages labeled $p_1, \ldots, p_5$. The links between these web pages are shown in Figure 1, where an arrow indicates that one page links to another. For example, page p_5 has links to pages p_1 and p_2.

For each web page p_i, the PageRank algorithm determines a *rank* r_i, which will be a numerical value indicating its importance (higher ranks being more important). This rank is based on two quantities we define for each page p_i: the number of links page p_i has to other pages, which we

[1] The term PageRank is trademark of Google; however, the patent for PageRank is held by Stanford University, which received shares of Google in exchange for the use of the patent. These shares were eventually sold for $336 million.

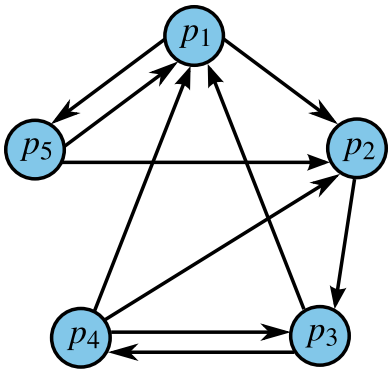

Figure 1: Example of a web having five pages.

denote by ℓ_i, and the set of other pages linking to page p_i, which we denote by B_i. For example, in Figure 1, $\ell_1 = 2$ and $B_1 = \{p_3, p_4, p_5\}$. Using these quantities, the rank of each page p_i is then given by the formula

$$r_i = (1-d) \; + \; d \sum_{p_j \in B_i} \frac{r_j}{\ell_j} \, , \tag{1}$$

where d is a design parameter chosen between 0 and 1. For example, upon applying this formula (1) to web page p_1 in Figure 1 when $d = 0.9$, we obtain $r_1 = 0.1 + .45r_3 + 0.3r_4 + 0.45r_5$.

We can think of the formula in (1) as follows: each page equally divides its own rank into ℓ_i fractions and passes one fraction to each of the ℓ_i pages it links to. A page then determines its own rank by summing up the fractions it receives from its B_i neighbors and using this value in equation (1). At first this may seem circular, because to calculate any page's rank, we first need to known the ranks of the other pages. However, we will next formulate this as a linear algebra problem and show how to find its solution.

The key idea here is that we need to *simultaneously* satisfy equation (1) for each web page. For the example in Figure 1 with $d = 0.9$, this results in the following system of linear equations:

$$\begin{aligned}
r_1 &= 0.1 && && + 0.45r_3 + 0.3r_4 + 0.45r_5 \\
r_2 &= 0.1 + 0.45r_1 && && + 0.3r_4 + 0.45r_5 \\
r_3 &= 0.1 && + 0.9r_2 && + 0.3r_4 \\
r_4 &= 0.1 && && + 0.45r_3 \\
r_5 &= 0.1 + 0.45r_1 \, . && &&
\end{aligned}$$

The ranks of the web pages are then given by the solution of this linear system.

We can use matrix/vector notation to express these relations in the more compact form

$$\mathbf{r} = (1-d)\mathbf{1} + dH\mathbf{r} \, , \tag{2}$$

where

$$H = \begin{bmatrix} 0 & 0 & 1/2 & 1/3 & 1/2 \\ 1/2 & 0 & 0 & 1/3 & 1/2 \\ 0 & 1 & 0 & 1/3 & 0 \\ 0 & 0 & 1/2 & 0 & 0 \\ 1/2 & 0 & 0 & 0 & 0 \end{bmatrix}, \qquad \mathbf{r} = \begin{bmatrix} r_1 \\ r_2 \\ r_3 \\ r_4 \\ r_5 \end{bmatrix},$$

and $\mathbf{1}$ is a 5×1 vector of all ones. Here, H is sometimes called the *hyperlink matrix* and $\mathbf{r}$ is the vector of page ranks. Note that the hyperlink matrix completely summarizes the links shown in Figure 1, i.e., if you were given this matrix you could recreate the set of links. Letting I denote the 5×5 identity matrix

$$I = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix},$$

we can rewrite (2) as follows:

$$I\mathbf{r} = (1-d)\mathbf{1} + dH\mathbf{r}$$

or equivalently

$$(I - dH)\mathbf{r} = (1-d)\mathbf{1}\,.$$

Assuming that the matrix $(I - dH)$ has an inverse, the solution to this system of equations is

$$\mathbf{r} = (I - dH)^{-1}(1-d)\mathbf{1}\,.$$

Using MATLAB, we can compute the solution $\mathbf{r}$ for the parameter choice $d = 0.9$ as follows:

```
>> H = [0   0   1/2 1/3 1/2;
        1/2 0   0   1/3 1/2;
        0   1   0   1/3 0  ;
        0   0   1/2 0   0  ;
        1/2 0   0   0   0   ];
>> I = eye(size(H));
>> one = ones(5,1);
>> d = 0.9;
>> r = (I-d*H)\((1-d)*one);

r =

    1.1961
    1.1352
    1.3314
    0.6991
    0.6382
```

This shows that web page p_3 has the highest rank, followed by p_1 and then p_2.

This example can be directly generalized to an arbitrary collection of n web pages, $p_1, \ldots, p_n$. The links between these web pages can again be summarized in a hyperlink matrix H. This will be a $n \times n$ matrix in which the entry in the i^{th} row and the j^{th} column is given by

$$H_{ij} = \begin{cases} 1/\ell_j, & \text{if } p_j \in B_i \\ 0, & \text{otherwise.} \end{cases}$$

Furthermore, the vectors $\mathbf{r}$ and $\mathbf{1}$ will now be vectors of n elements.

Once you have understood this basic version of PageRank together with the MATLAB code above, think about answers to the following questions:

1. Assuming each page has at least one outgoing link, the sum of the elements of the solution vector $\mathbf{r}$ will always be n, the total number of pages. In other words, the average rank will always be exactly 1. Why?

2. The *adjacency matrix* for a web containing n pages is an $n \times n$ matrix A whose entry in row i and column j is 1 if page p_i has a link to page p_j, and 0 otherwise. How could you write MATLAB code to calculate the hyperlink matrix H given an adjacency matrix A?

3. Do you think the choice of the parameter d will affect the ranking for a given web? How could you use MATLAB to find out?

3 Iterative Algorithms

When applying the PageRank algorithm to the actual web, the number of pages involved can be quite large. Moreover, even if n is only a modest 10 million, then H will have $n^2 = 10^{14}$ entries! Fortunately, most of these entries are zero, as a typical web page only links to tens or hundreds of others, not thousands or millions. For example, for this n, if each page links to a hundred others on average, then only 0.001% of the entries of H will be nonzero. Matrices having mostly zero entries are called *sparse matrices*, and it is often more efficient to solve sparse systems of linear equations by using an an *iterative algorithm* instead of the "left division" algorithm from Section 2. The main idea behind an iterative algorithm is to have a procedure for generating a sequence $\mathbf{r}(0), \mathbf{r}(1), \mathbf{r}(2), \ldots$ of guesses to the solution of the linear system, where at each step the previous guess is used to help make a new (and hopefully better) guess. We illustrate such a procedure by using the relation in (2) to generate new guesses at each step. Specifically, given the k^{th} guess $\mathbf{r}(k)$, we let the $(k+1)^{\text{st}}$ guess be the left-hand side of (2) when the previous guess is substituted into the right-hand side, i.e.,

$$\mathbf{r}(k+1) = (1-d)\mathbf{1} + dH\mathbf{r}(k)\,.$$

Note that if $\mathbf{r}(k)$ was a solution to the linear system, then it must be that $\mathbf{r}(k+1) = \mathbf{r}(k)$, i.e., if a guess is a solution, then our rule will not change the guess.

We are interested in what happens when the guess is not a solution. To see what happens in this case, let us apply this procedure to the 5-node example in Figure 1 starting with an initial guess of $\mathbf{r}(0) = \mathbf{1} = [1 \;\; 1 \;\; 1 \;\; 1 \;\; 1]^{\mathrm{T}}$. To generate a sequence of guesses we can use the following simple M-file, which we call `guess.m`:

```
r = one;
fprintf('Step  0:'); fprintf('%10.4f', r); fprintf('\n');
for k = 1:15
  r = (1-d)*one + d*H*r;
  fprintf('Step %2u:', k); fprintf('%10.4f', r); fprintf('\n');
end;
```

Running this M-file creates the following output, assuming the variables one, H and d are already defined in the workspace as above:

```
>> guess
Step  0:    1.0000    1.0000    1.0000    1.0000    1.0000
Step  1:    1.3000    1.3000    1.3000    0.5500    0.5500
Step  2:    1.0975    1.0975    1.4350    0.6850    0.6850
Step  3:    1.2595    1.1076    1.2933    0.7458    0.5939
Step  4:    1.1729    1.1577    1.3206    0.6820    0.6668
Step  5:    1.1989    1.1325    1.3466    0.6943    0.6278
Step  6:    1.1967    1.1303    1.3275    0.7060    0.6395
Step  7:    1.1969    1.1381    1.3291    0.6974    0.6385
Step  8:    1.1946    1.1352    1.3335    0.6981    0.6386
Step  9:    1.1969    1.1344    1.3311    0.7001    0.6376
Step 10:    1.1959    1.1355    1.3310    0.6990    0.6386
Step 11:    1.1960    1.1352    1.3317    0.6989    0.6382
Step 12:    1.1961    1.1351    1.3314    0.6993    0.6382
Step 13:    1.1961    1.1352    1.3313    0.6991    0.6382
Step 14:    1.1960    1.1352    1.3314    0.6991    0.6382
Step 15:    1.1961    1.1352    1.3314    0.6991    0.6382
```

The guesses generated in this manner are indeed approaching the solution we found directly in the previous section. Moreover, if we only care about the order of the ranks, then after only $k = 3$ iterations, this procedure has generated the correct order.

The above example raises the following two questions:

1. Does this method converge to the correct solution for any set of web pages?

2. Does the choice of initial vector $\mathbf{r}(0)$ matter?

In short, the answer to these is that, provided the parameter d is chosen so that $0 < d < 1$, the method will converge to the correct solution for any initial choice of $\mathbf{r}$. Moreover, by choosing d to be around 0.85, Brin and Page report that the calculation of $\mathbf{r}$ for the entire World Wide Web can be done in about 50–100 iterations.