

Google Eigenvectors

Gerald Farin, Dianne Hansford

October 23, 2005

This note is about search engines. While many search engine techniques are highly proprietary, they all share the basic idea of *ranking* web pages. The concept was first introduced by Prim and Page in 1998, and forms the basics of the search engine Google. Here we will show you how ranking web pages is essentially a straightforward eigenvalue problem!

The web (at some frozen point in time) consists of N web pages, most of them pointing to (having links to) other web pages. A page which is pointed to very often would be considered important, while a page with none or only very few other pages pointing to it would be considered not important. How can we rank all web pages according to how important they are? In the sequel, we assume all web pages are ordered in some fashion (such as lexicographic) so we can assign a number, such as i to any page.

First, two definitions: if page i points to page j , then we say that page i has page j as an *outlink*, whereas if page j points to page i , then page i has page j as an *inlink*. A page is not supposed to link to itself.

We can represent the connectivity structure of the web by an $N \times N$ matrix C : if page i points to page j , then $c_{i,j} = 1$, if it does not, then $c_{i,j} = 0$. An example matrix is the following:

$$C = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}.$$

In this example, page 1 has two outlinks and one inlink.

The ranking r_i of any page i is entirely defined by C . Here are some rules (with increasing sophistication) for ranking page i :

1. The ranking r_i should grow with the number of page i 's inlinks.
2. The ranking r_i should be weighted by the ranking of each of page i 's inlinks.
3. Let page i have an inlink from page j . Then the more outlinks page j has, the less it should contribute to r_i .

Some comments. Rule 1: a page which is pointed to very often should deserve high ranking.

But, rule 2: if all those inlinks prove to be low-ranked, then their sheer number is mitigated by their low rankings. Conversely, if they are mostly high-ranked, then they should boost page i 's ranking.

Rule 3: If page j has only one outlink, and it points to page i , then page i should be "honored" for such trust from page j . Conversely, if page j points to a large number of pages, page i among them, this does not give page i much pedigree.

Now we will create a formal framework for these three rules. Observe that the inlinks of page i are described by the i^{th} column of C , and its outlinks are described by row i .

Let o_j the total number of outlinks of page j . This is simply the sum of all elements of the j^{th} column of C . The more outlinks page j has, the lower its contribution to page i 's ranking it will have. Thus we scale every element of column j by $1/o_j$. We call the resulting matrix D with

$$d_{i,j} = \frac{c_{i,j}}{o_j}.$$

Note that all columns of D have nonnegative entries and sum to one. Matrices with that property are called *stochastic*.

In our example above, we have

$$D = \begin{bmatrix} 0 & 1/2 & 1/2 \\ 1 & 0 & 1/2 \\ 0 & 1/2 & 0 \end{bmatrix}.$$

How do rules 1-3 translate into computable equations? We first try rule 1. It would suggest to set

$$r_i = \sum_{j=1}^N c_{i,j}; \quad i = 1, \dots, N.$$

Bringing in rule 2, we refine this to

$$r_i = \sum_{j=1}^N c_{i,j} r_j; \quad i = 1, \dots, N.$$

Refining this again, using rule 3, we get

$$r_i = \sum_{j=1}^N d_{i,j} r_j; \quad i = 1, \dots, N.$$

This begs for being written in matrix form! Using the vector $\mathbf{r} = [r_1, \dots, r_N]^T$, we have

$$\mathbf{r} = D\mathbf{r}. \tag{1}$$

This states that we are looking for the eigenvector of D corresponding to the eigenvalue 1. But: how do we know that D does indeed have an eigenvector 1? The answer is that all stochastic matrices have that property and thus no trouble arises. (No proof here, though).

In order to find \mathbf{r} , we simply employ the power method from Section 16.4. This method needs an initial guess for \mathbf{r} , and setting all $r_i = 1$ is not too bad for that. As the iterations converge, the solution is found. The entries of \mathbf{r} are real, since they correspond to a real eigenvalue.

The vector \mathbf{r} now contains the ranking – called page rank by Google – of every page. If Google retrieves a set of pages all containing a link to a term you are searching for, it presents them to you in decreasing order of the pages' ranking.

An example: let C be given by

$$C = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 \end{bmatrix}.$$

Thus

$$D = \begin{bmatrix} 0 & 0 & 0 & 1/2 & 0 & 1/3 \\ 1/3 & 0 & 1/2 & 1/2 & 0 & 1/3 \\ 0 & 0 & 0 & 0 & 1/3 & 1/3 \\ 1/3 & 0 & 0 & 0 & 1/3 & 0 \\ 0 & 0 & 1/2 & 0 & 0 & 0 \\ 1/3 & 1 & 0 & 0 & 1/3 & 0 \end{bmatrix}.$$

The eigenvector corresponding to the eigenvalue 1 is given by

$$\mathbf{r}^T = [0.306, 0.548, 0.278, 0.148, 0.139, 0.697]$$

Thus page 6 has the highest ranking and page 5 has the lowest ranking.

Some final remarks: Google (the company) solves equation (1) about once a month.¹ Roughly speaking, they do not use $r_i = 1$ as the initial guess, but instead use last month's solution.

Final remark: in the real world, $N \approx 10,000,000,000 = 10^{10}$, meaning that D has 10^{20} elements. This is world's largest matrix to be used ever. Luckily, it contains

¹The actual equation is a bit trickier, and is omitted here.

mostly zeroes and thus is extremely sparse. Without taking advantage of that, Google (and other search engines) could not function.

Summary: we have seen how a basic web problem can be solved by observing that its intrinsic structure is that of an eigenvalue problem.