The PageRank Algorithm

Huck Bennett and Aleks Chakarov

April 27, 2011

1 Introduction

Since Larry Page and Sergey Brin founded Google in the late nineties, the need for a good internet search engine has become obvious. In America and much of the world, Google has by far the highest market share of search engines. A significant part of what helped to establish both Google's technical superiority and its financial success came from the PageRank algorithm (named after Larry Page, not web pages) for ranking the importance of web pages. These notes give the basis for the algorithm as well as analysis and extensions.

We may consider numerous statistics for ranking webpages including:

- The number of views that a webpage receives per day.
- The number of webpages that point to it.
- The number of important webpages that point to it.

The PageRank algorithm formalizes this last measure, using a recursive definition for which pages are "important" relative to the remaining pages, as we will discuss later.

We envision the world wide web as a graph $G = (V, E)$ where the vertices V correspond to websites and the directed edges e_{ij} correspond to the existence of a link from page i to page j.

For much of the lecture we will follow the notation given in [1].

2 A Ranking Metric

We introduce some terminology:

- Let $I(P_i)$ denote the cardinal rank (importance) of page i.
- Let B_i, F_i denote the sets of pages pointing and eminating from page i respectively.
- Let $\ell_i = |F_i|$ be the number of edges eminating from page i.

We will first define a naive metric over webpages to build intuition:

$$I(P_i) = \sum_{P_j \in B_i} \frac{I(P_j)}{\ell_j} \tag{1}$$

Equation 1 declares that the rank of a page P_i is equal to the sum of the ranks of all pages P_j that point to P_i, normalized by the number of outgoing links these pages P_j have.

Example

Suppose that only two pages P_{j_1} and P_{j_2} point to P_i. Furthermore, suppose that $I(P_{j_1}) = 1, \ell_{j_1} = 1, I(P_{j_2}) = 10, \ell_{j_2} = 5$. Then $I(P_i) = \frac{I(P_{j_1})}{\ell_{j_1}} + \frac{I(P_{j_2})}{\ell_{j_2}} = \frac{1}{1} + \frac{10}{5} = 3$.

This example illustrates a problem, however. Once we had the ranks for P_{j_1} and P_{j_2} we were able to compute the rank of P_i, but when we start analyzing a network we presumably don't know the rank of any of the pages – equation 1 only defines ranks recursively. We therefore cannot compute the ranks directly, and must rely instead on an iterative technique. We will consolidate our rank valuations of pages into rank vectors $\vec{r}$ where the ith component of $\vec{r}$ represents the rank $I(P_i)$ of page P_i. We will start with an initial uniform rank vector $\vec{r_0}$, and a "hyperlink matrix" as described below:

Define the matrix

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

Consider the (initially unweighted) adjacency matrix of our network, with the entries divided by ℓ_i. The "hyperlink" matrix is simply the transpose of this normalized adjacency matrix. We will see the motivation for this definition shortly. We may give this matrix a probabilistic interpretation, and will see that it is similar to the transititon matrix of a Markov chain.

3 A Short Linear Algebra Review

Recall that given an $n \times n$ matrix B, a vector $\vec{v}$, and a scalar λ that if $B\vec{v} = \lambda\vec{v}$ then $\vec{v}$ is called an **eigenvector** of B and λ an **eigenvalue** of B. We call the eigenvector associated with the eigenvalue of largest magnitude the **dominant eigenvector** of B.

We define some matrix regularity conditions:

- A matrix is **stochastic** if all of its entries are nonnegative, and if the elements in each column sum to 1. This will ensure the existence of a stationary vector.

- A matrix B is **primitive** if for some $m \geq 0$ we have that B^m has all positive entries. This property guarantees that $|\lambda_2| < 1$, where λ_2 is the eigenvector with second largest magnitude. We need this to ensure convergence of the power method (described below).

- A matrix B is **irreducible** if its underlying adjacency matrix yields a strongly connected graph. We need this final condition to ensure that the stationary vector $\vec{r}^*$ has all positive entries.

We have an iterative procedure called the **power method** for computing the rank vector $\vec{r}^*$: given an initial rank vector $\vec{r_0}$, for example the uniform distribution, we compute $\vec{r_n} = H\vec{r}_{n-1}$. By repeatedly applying H the sequence $\vec{r_0}, \vec{r_1}, \vec{r_2}, \ldots$ will converge to the dominant eigenvector $\vec{r}^*$, which in this case has corresponding eigenvalue 1 (since H is assumed to be stochastic).

If we apply a matrix with these regularity properties iteratively to a given initial rank distribution $\vec{r_0}$ then eventually our sequence $\vec{r_0}, \vec{r_1}, \vec{r_2}, \ldots$ will converge to the dominant eigenvector $\vec{r}^*$.

We are therefore looking for a fixed point of the operator H: we want to find $\vec{r}^*$ satisfying the condition $H\vec{r}^* = \vec{r}^*$.

The **one norm** of a vector $\vec{v}$ is defined as $||\vec{v}||_1 = \sum_{i=1}^{n} |v_i|$.

We can measure progress toward our goal by computing the change between iterations of the algorithm, measured in terms of the one norm: $\delta_k = ||\vec{r_k} - \vec{r}_{k-1}||_1$, which denotes the amount of change incurred from the kth application. When $\delta_k < \epsilon$ for some initially fixed ϵ we terminate the procedure. We summarize this in the algorithm given below:

4 The Basic Algorithm

Algorithm: Naive PageRank

Input: Initial rank vector $\vec{r_0}$, inverted weighted adjacency matrix H, convergence threshold ϵ

Output: Sufficiently optimized vector $\vec{r_k}$

$\delta = \infty$

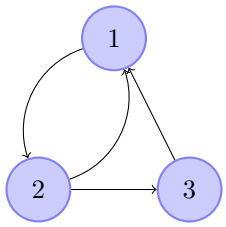
while $\delta > \epsilon$ **do**

  $\vec{r}_{i+1} = H\vec{r_i}$

  $\delta = ||\vec{r}_{i+1} - \vec{r_i}||_1$

end

Example



The hyperlink matrix of this network is

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

If we start with

$$\vec{r_0} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

and iterate the procedure until we converge, we obtain the following results:

$\vec{r_0}$	$\vec{r_1}$	$\vec{r_2}$	$\vec{r_3}$	$\vec{r_4}$	$\vec{r_5}$	$\vec{r_6}$	$\cdots$	$\vec{r}_{15}$	$\cdots$	$\vec{r}^*$
1	0	.5	.5	.25	.5	.375	$\cdots$	.39844	$\cdots$	.4
0	1	0	.5	.5	.25	.5	$\cdots$	.39844	$\cdots$	.4
0	0	.5	0	.25	.25	.125	$\cdots$	.20312	$\cdots$	.2

We see that by the 15th iteration we are fairly close to converging: $||\vec{r}^* - \vec{r}_{15}||_1 = .00624$.

5 Rank Sinks

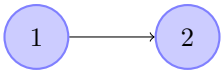
Similar to the three questions posed in [1], we want to ensure that our method has the following properties:

- The sequence $\vec{r_0}, \vec{r_1}, \ldots$ always converges.
- The sequence $\vec{r_0}, \vec{r_1}, \ldots$ always converges to the same vector $\vec{r}^*$ regardless of the starting vector $\vec{r_0}$.
- Rank weight is conserved and every page P_i satisfies $I(P_i) > 0$.

As we will see, if our matrix has all of the desirable conditions given in the linear algebra section, all of these properties will hold.

Consider the following straightforward network, as given in [1]:

Example



The hyperlink matrix of this network is

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

Assuming that we start with initial distribution

$$\vec{r_0} = \begin{bmatrix} 1/2 \\ 1/2 \end{bmatrix}$$

and repeatedly compute $\vec{r_k} = H\vec{r}_{k-1}$. We get the sequence:

$\vec{r_0}$	$\vec{r_1}$	$\vec{r_2}$
1/2	0	0
1/2	1/2	0

Here the rank vectors "converge" to $\vec{0}$, the zero vector. In other words, rank was not conserved since vertex 2 forms a **rank sink** – it does not have a link going back to 1. In order to remedy this flaw in the naive method we introduce a new matrix S which is equal to H except each $\vec{0}$ column in H is replaced with the uniform distribution $(1/n)\vec{1}$. Let A be the matrix which has all zero columns except for the $\vec{0}$ columns corresponding to dangling nodes, which are set to the uniform distribution [1]. In other words, $A = \begin{bmatrix} 0 & 1/2 \\ 0 & 1/2 \end{bmatrix}$ and so $S = H + A$.

We take another looks at the previous network with this modification. For the above graph we have that

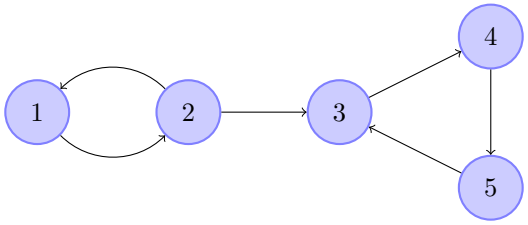
$$S = \begin{bmatrix} 0 & 1/2 \\ 1 & 1/2 \end{bmatrix}$$

giving the new sequence

$\vec{r_0}$	$\vec{r_1}$	$\vec{r_2}$	$\vec{r_3}$	$\cdots$	$\vec{r}_{10}$	$\cdots$	$\vec{r}^*$
.5	.25	.375	.3125	$\cdots$	.333496	$\cdots$	.333333
.5	.75	.625	.6875	$\cdots$	.666504	$\cdots$	.666667

that converges to the point where $I(P_1) = 1/3, I(P_2) = 2/3$, giving that P_2 has twice the rank of P_1. We see that even in the first 10 iterations we get that $||r_{10} - r^*||_1 \approx 3.26 \cdot 10^{-4}$, demonstrating fast, correct convergence of the new technique. A problem with rank sinks remains, however:

Example



We get that the network shown above has

$$S = H = \begin{bmatrix} 0 & 1/2 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1/2 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

where if we begin with $\vec{r_0} = \begin{bmatrix} 0 \\ 0 \\ 1/3 \\ 1/3 \\ 1/3 \end{bmatrix}$ converges to $\vec{r}^* = \begin{bmatrix} 0 \\ 0 \\ 1/3 \\ 1/3 \\ 1/3 \end{bmatrix}$.

However, if we change the initial distribution $\vec{r_0}$, the result towards which we converge varies and in certain cases ($\vec{r_0} = [0\ 0\ 1\ 0\ 0]^T$) even fails to converge.

Even though none of the nodes in this network are dangling, vertices $3 \to 4 \to 5$ form a rank sink cycle – pages P_1 and P_2 ultimately retain no rank. We need to make one final modification to our iteration matrix to ensure that our network is strongly connected.

6 The Full PageRank Algorithm

Our model so far assumes that when a web user has the option of clicking through to another website, he does so by randomly choosing one of the outgoing hyperlinks, and when there are none for some page then he decides to moves to a random website. However, intuitively, a user may move to a random website at any time, even if presented with outgoing hyperlinks. The papers [3] and [1] describe the "random surfer" model: with probability α a user follows through a hiperlink to another page, and with probability $1 - \alpha$ he moves to a random page. This furthers the interpretation that the rank of a page indicates the

"probability" that a standard user will be at a particular page, or alternatively it signifies the fraction of time a user is expected to spend at that page. This gives us the following new transition matrix, called the 'Google' matrix G in [1]:

$$G = \alpha S + (1 - \alpha)\mathbf{1} \tag{2}$$

Where $\mathbf{1}$ indicates the $n \times n$ matrix filled entirely with 1's. Page et al. in [3] suggest that $\alpha = .85$ is a good choice for α, although any $0 < \alpha < 1$ suffices. We generally assume that the random surfing component is uniform at random, though this need not be the case; [3] discusses different weightings.

Algorithm: Full PageRank

Input: Initial rank vector $\vec{r_0}$, 'Google' matrix G, convergence threshold ϵ
Output: Sufficiently optimized vector $\vec{r_k}$

$\delta = \infty$
while $\delta > \epsilon$ **do**
  $\vec{r}_{i+1} = G\vec{r_i}$
  $\delta = ||\vec{r}_{i+1} - \vec{r_i}||_1$
end

Notice that the steps in the algorithm are exactly the same as the naive algorithm: the only difference is what iteration matrix we use. In this case we use G instead of H.

7 Time Analysis

We first present basic time analysis. Assuming that we are computing the ranks of a total of n webpages, each of the vectors in the above algorithm has size n, and G is an $n \times n$ matrix. Within the 'while' loop there are two operations: matrix-vector multiplication and distance finding. Computing $\vec{r}_{i+1}$ and G involves taking the dot product of n vectors of length n, and hence takes $O(n^2)$ time. Updating δ involves componentwise subtraction and addition over two vectors of length n, and so takes $O(n)$ time. Therefore, each time through the loop takes $O(n^2)$ time.

However, we can often do much better. Following the analysis in [1], recall that $G = \alpha H + \alpha A + \frac{1-\alpha}{n}\mathbf{1}$ so $G\vec{r_k} = \alpha H\vec{r_k} + \alpha A\vec{r_k} + \frac{1-\alpha}{n}\mathbf{1}\vec{r_k}$. In general, H is sparse and all of the rows of A and $\mathbf{1}$ are the same. We can take advantage of this structure: because of the uniformity of their rows, we need only compute the dot product of one row each of A and $\mathbf{1}$ with $\vec{r_k}$, for a total of $O(n)$ computations. Furthermore, suppose that H has only a constant number ($O(1)$) non-zero entries per column ([1] suggests empirically that this number is often around 10). Then we need only perform $O(n) \cdot O(1) = O(n)$ multiplications between H and $\vec{r_k}$. Thus under these assumptions an iteration of the process is reduced from taking $O(n^2)$ to $O(n)$ time. The bottleneck stems from the multiplication of H and $\vec{r_k}$, so if we can make a sparseness assumption about the

number $f(n) = o(n)$ of non-zero entries per column in H, then we can conclude that an iteration takes $O(nf(n))$ time.

Of course, the total running time of the algorithm is also heavily dependent on ϵ, however it is unclear how fast our convergence is in general. As discussed in [1], the convergence of the power method depends on the size of the magnitude of the second largest eigenvalue of the 'Google' matrix (often called the spectral gap of the matrix in the literature) – the larger the spectral gap the faster the method converges. Experiments from [3] show that the algorithm converges exponentially fast in practice. For generality we will denote the running time of the entire algorithm as $O(nf(n)g(\epsilon))$ for some function g.

8 Implementation

9 Conclusion

We have given a thorough treatment of the intuition and basic model behind PageRank, culminating in a robust algorithm for computing the relative importance of web pages. Although Google has moved on to more sophisticated techniques for its general ranking of web pages, PageRank still appears in many places, such as the Google Toolbar [4].

PageRank has also been suggested as a tool for analysis in academia [4]. A journal's "eigenfactor" (i.e. impact factor computed via PageRank) may be computed by forming a graph where each journal forms a vertex and each citation between journals forms an edge. Another application involves universities' placement of their Ph.D. graduates at other universities. In this case each university forms a vertex and each placement at another university an edge.

Further applications include analysis of protein networks, ecosystems, street networks, and many other applications which involve networks.

References

[1] Austin, David. "How Google Finds Your Needle in the Web's Haystack." Feature Column from the AMS, 2011.

[2] Bryan, Kurt and Tanya Leise. The "$25,000,000,000 eigenvector and the linear algebra behind Google." Preprint, 2008.

[3] Page, Lawrence; Brin, Sergey; Motwani, Rajeev and Terry Winograd. "The PageRank Citation Ranking: Bringing Order to the Web." Technical Report, Stanford InfoLab, 1999.

[4] http://wikipedia.org/wiki/PageRank