MATH 51 LECTURE NOTES: HOW GOOGLE RANKS WEB PAGES

BRIAN WHITE

During a typical use of a search engine,

(1) the user types a word,
(2) the engine finds all web pages that contain the given word, and
(3) the engine lists the pages in some order, ideally from most interesting to least interesting.

When Google was launched, there were already a number of search engines. Google's main innovation was a superior way of doing step 3, that is, a better way of ranking web pages. Google's method is called the PageRank algorithm and was developed by Google founders Sergey Brin and Larry Page while they were graduate students at Stanford. At that time, Brin and Page were 23 and 24 years old, respectively.

In the PageRank method, the ranking of pages is based solely on how pages are linked and not on the content of the pages or on how often the pages are visited. Of course the web is constantly changing, so the rankings change, too. Google calculates the ranks once a month. Each time you do a search, Google gives a list of the relevant pages in order of that month's ranking.

The rankings form the entries of a vector $\mathbf{v} \in \mathbf{R}^N$, where N is the total number of pages on the web. The entries of $\mathbf{v}$ are nonnegative numbers, and v_i is supposed to be a measure of the value (or interest or importance) of page i. Thus in a given search, the first page Google will list is the page i (among those containing the search word) with the highest value of v_i.

In this note, we describe a very simple ranking vector $\mathbf{v}$, followed by several improvements leading to the PageRank method.

1. The Very Simple Method

If page j has a link to page i, imagine that page j is recommending page i. One could rank pages purely by the number of recommendations each page gets. Let N be the total number of web pages (over 4 billion, according to Google.) Let A be the N by N matrix whose ij entry is

$$a_{ij} = \begin{cases} 1 & \text{if page } j \text{ has a link to page } i \\ 0 & \text{if not.} \end{cases}$$

Then the number of recommendations that page i gets is

$$a_{i1} + a_{i2} + \cdots + a_{in}.$$

We can think of this as a measure of the "value" v_i of page i

$$v_i = a_{i1} + a_{i2} + \cdots + a_{in}. \tag{1}$$

Date: November, 2004.

This equation can be written very concisely using matrix multiplication:

$$\mathbf{v} = A\mathbf{u}$$

where $\mathbf{u} \in \mathbf{R}^N$ is the vector each of whose components is 1.

2. First improvement

To be included on a short list of recommended restaurants (for example) presumably means more than to be included on a long list. This suggests that the weight of a recommendation made by page j should be inversely proportional to the total number of recommendations that j makes. Thus we replace the formula (1) by:

$$v_i = \frac{a_{i1}}{n_1} + \frac{a_{i2}}{n_2} + \dots \frac{a_{iN}}{n_N}. \tag{2}$$

where n_j is the total number of recommendations that page j makes:

$$n_j = \sum_i a_{ij}.$$

There is a problem if page j is a "dead-end", i.e., if there are no links from page j, because then the formula (2) involves dividing by 0. Google gets around this problem by pretending that a page with no links out in fact has a link to *every* page, including itself. Thus we redefine the matrix A as follows:

$$a_{ij} = \begin{cases} 1 & \text{if page } j \text{ has a link to page } i, \text{ or if page } j \text{ is a dead end} \\ 0 & \text{if there is a link from page } j \text{ to some page, but not to page } i. \end{cases}$$

Now $n_j = \sum_i a_{ij}$ will always be positive, so formula (2) makes sense.

The formula (2) can be written more succinctly as

$$\mathbf{v} = P\mathbf{u}.$$

where P is the N by N matrix whose ij entry is

$$p_{ij} = \frac{a_{ij}}{n_j}$$

3. A Further Improvement

The weight of a recommendation should depend on the importance of the recommender. Since v_j is supposed to reflect the value or importance of page j, we thus multiply the j^{th} term in (2) by v_j to get the formula

$$v_i = \frac{a_{i1}}{n_1}v_1 + \frac{a_{i2}}{n_2}v_2 + \dots + \frac{a_{iN}}{n_N}v_N,$$

or in matrix notation:

$$\mathbf{v} = P\mathbf{v}.$$

Thus we are led to conclude that the ranking vector $\mathbf{v}$ should be an eigenvector of P with eigenvalue 1.

Note for this to give a ranking of the webpages, 1 had better be an eigenvalue of P. Fortunately, it must be because P is a Markov matrix. That is, the entries of P are all ≥ 0 and the sum of the entries in each column is 1.

Theorem 1. *Suppose P is a Markov matrix. Then $\lambda = 1$ is an eigenvalue of P. Furthermore, there is an eigenvector $\mathbf{v}$ with eigenvalue 1 and with each entry ≥ 0.*

Proof. Since the sum of the elements in each column of P is 1, the sum of the elements in each column of the matrix $I - P$ is 0. Thus if we add the first $N - 1$ rows of $I - P$ to the last row, we get a matrix whose last row is 0. This means that the rank of $I - P$ is at most $N - 1$, which means that the equation $(P - I)\mathbf{v} = \mathbf{0}$ must have at least one free variable, which means that the equation has a nonzero solution $\mathbf{v}$. Of course such a $\mathbf{v}$ is an eigenvector of P with eigenvalue 1.

The proof that there is an eigenvector with all entries ≥ 0 is more complicated and we omit it. □

The Markov matrix P may be regarded as describing a random walk over the web in which you move from page to page as follows. When you're on a given page, you choose at random one of the links from that page and then you follow that link. (For example, if there are six links, you roll a die to choose which link to follow.) If you're at a dead end (no links), you choose a page at random from the entire web and move to it. If we imagine a large population of web surfers moving around the web in this way, then p_{ij} gives the expected fraction of those surfers on page j who will then move to page i. A solution $\mathbf{v}$ to the equation $\mathbf{v} = P\mathbf{v}$ can be regarded as describing an equilibrium or steady state distribution of surfers on the various pages.

4. A Final Improvement

One last improvement will give us the PageRank method.

According to theorem 1, the matrix P does have an eigenvector with eigenvalue 1. But if there are several clusters of webpages not connected to each other, then P will have a set of two or more linearly independent eigenvectors with eigenvalue 1. In that case, the equation $\mathbf{v} = P\mathbf{v}$ will not determine a unique ranking.

For example, suppose that there are only four pages. Suppose that pages 1 and 2 link to each other, and that pages 3 and 4 link to each other, but that there are no links from pages 1 or 2 to pages 3 or 4 or vice versa. The equation $\mathbf{v} = P\mathbf{v}$ implies that $v_1 = v_2$ and that $v_3 = v_4$, but it gives no information about the relative sizes of v_1 and v_3.

Another problem with $\mathbf{v} = P\mathbf{v}$ is that many pages are likely to be tied with a score of 0. Indeed, if it is possible by following links to get from page j to page i but not vice versa, then v_j turns out to be 0.

Google avoids these problems by replacing P by another Markov matrix Q. Let us first describe the random web walk given by Q.

Fix some parameter r with $0 < r < 1$. (Google uses $r = .85$.) Imagine you have a biased coin, so that the probability of heads is r. (Of course if $r = 1/2$, then it is a fair coin.) Now move around the web as follows. If you're on page j, flip the coin. If you get heads, then choose randomly one of the links from page j and follow that link. If page j is a dead-end, or if the coin comes up tails, then pick a page at random from the whole web and jump (or "teleport") to that page.

In mathematical notation, we let

$$Q = rP + (1 - r)T$$

where T is the N by N "teleportation" matrix, i.e, the matrix each of whose entries is $1/N$.

Note that Q is still a Markov matrix. However, unlike P, the matrix Q has no zeros. Thus we can apply the following theorem to Q:

Theorem 2. *Let Q be a Markov matrix none of whose entries is 0. Then there is a unique vector $\mathbf{v}$ such that*

(1) *The entries of $\mathbf{v}$ are all positive: $v_i > 0$ for all i.*
(2) *The sum of the entries is 1: $\sum_i v_i = 1$.*
(3) $\mathbf{v} = Q\mathbf{v}$.

Furthermore, if $\mathbf{w}$ is any vector satisfying condition 2 ($\sum_i w_i = 1$), then $Q^k\mathbf{w}$ converges to $\mathbf{v}$ as $k \to \infty$.

Thus, up to scaling, Q has a unique eigenvector $\mathbf{v}$ with eigenvalue 1. We find that eigenvector, normalize it so that the sum of the entries is 1. Then we use $\mathbf{v}$ to rank the webpages: the page i with the largest v_i is ranked first, and so on.

5. How to Find $\mathbf{v}$

In general, linear systems are solved by Gaussian elimination. But there are over four billion web pages, so solving $\mathbf{v} = Q\mathbf{v}$ (or $(Q - I)\mathbf{v} = \mathbf{0}$) by Gaussian elimination takes too long. Indeed, for a computer doing a trillion multiplications per second it would take about a billion years to solve this system by Gaussian elimination.

Fortunately, theorem 2 gives another method. We begin with any nonzero vector whose entries are all nonnegative. It is convenient to divide this vector by the sum of its entries to get a vector $\mathbf{w}$. Of course the sum of the entries of $\mathbf{w}$ will then be 1. We repeatedly multiply by Q to get $Q\mathbf{w}, Q^2\mathbf{w}, Q^3\mathbf{w}$, and so on. By theorem ?, these vectors converge to $\mathbf{v}$. So we keep going until the sequence settles down, i.e. until $Q^{k+1}\mathbf{w}$ is nearly equal to $Q^k\mathbf{w}$. Then $Q^k\mathbf{w}$ is (for all practical purposes) our eigenvector $\mathbf{v}$.

Even just multiplying a huge vector by a huge matrix could take a while. But for the particular matrix Q, we can find $Q\mathbf{w}$ much more easily than it might first appear. Note that

$$Q\mathbf{w} = rP\mathbf{w} + (1 - r)T\mathbf{w}.$$

Note that each entry of $T\mathbf{w}$ is just the average of the entries of $\mathbf{w}$. Since $\mathbf{w}$ has N entries that add up to 1, the average is $1/N$. Thus $T\mathbf{w} = \mathbf{u}/N$, so

$$Q\mathbf{w} = rP\mathbf{w} + \frac{1 - r}{N}\mathbf{u}.$$

Note also that the multiplication $P\mathbf{w}$ can be done rather quickly because the vast majority of the entries of P are 0.

Thus calculating the eigenvector $\mathbf{v}$ takes Google days rather than eons.

References

[1] S. Kamvar, T. Haveliwala, and G. Golub, *Adaptive methods for the computation of PageRank*, Linear Algebra and its Applications, 2004, pp. 51–65. http://www.stanford.edu/~taherh/papers

[2] L. Page, S. Brin, R. Motwani, and T. Winograd, *The PageRank citation ranking: bringing order to the web*, Technical Report, Stanford Digital Libraries Project, 1998. http://dbpubs.stanford.edu:8090/pub/1999-66.

[3] D. Higham and A. Taylor, *The sleekest link algorithm*, http://www.maths.strath.ac.uk/ãas96106/rep20_2003.pdf.

[4] T. Haveliwala and S. Kamkar, *The second eigenvalue of the google matrix*, Stanford University Technical Report, March 2003, http://dbpubs.stanford.edu:8090/pub/2003-20