APPM 3310: Three Methods of Computing the Stability Distribution of a Markov Transition Matrix Similar to PageRank

Kanoa Mignard, Aaron Li, and John Montagu

December 10 2019

1 Abstract

PageRank and similar algorithms are indispensable tools for finding relevant and useful websites on the world wide web, which contains billions of pages. Utilizing properties of regular transition matrices, we formulated three methods to compute the stability distribution containing the ranks of n pages in a web. The first two require unrealistic assumptions and are unfeasible to compute with large webs, but provide valuable insight into what the stability distribution represents and how Google determines the rank of a page. The third method makes reasonable assumptions and can be practically computed even with billions of pages.

2 Attribution

Kanoa wrote the Abstract, Introduction, and Mathematical Formulation. Aaron made all the code and wrote the Numerical Examples. John wrote the Discussion and Conclusions and made the slides for the presentation.

3 Introduction

Google's PageRank algorithm takes advantage of properties of regular Markov transition matrices to assemble a profile of the significance of every page on the internet. The internet has well over a billion websites, making the prospect of attempting to find useful pages without such a ranking system neigh impossible; however, computers have computational limits so Google exploits the laws of linear algebra to actualize their ranking system. In the following sections, we examine the behavior of a toy model of PageRank on a small-scale web, but these behaviors can be scaled up to arbitrary size due to the properties of Markov transition matrices laid out here.

PageRank identifies every URL as a unique page. Every page has at least zero outgoing links to other pages and incoming links from other pages to itself. Repeated links to any page are ignored and links to itself are ignored. The key assumption of PageRank is that every link on a page has an equal probability of being selected by a user called a random surfer. In the simplest case we will also assume that all pages link to at least one other page and are linked to by at least one other page, and that surfers cannot 'teleport', i.e. manually enter a URL. While we will always preserve the first assumption, we will eventually relax the other assumptions by introducing a damping factor to make an irregular transition matrix regular. In the broadest sense, pages of a web that have the most incoming links from and outgoing links to other pages will have the highest rank.

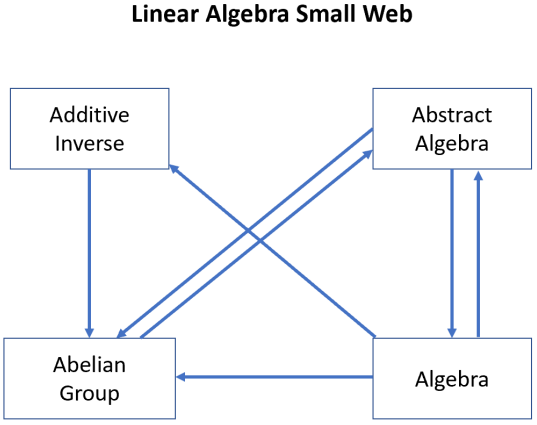

Fig 3.1: Web Created From Links to Wikipedia's Linear Algebra Page

Given a web such as figure 3.1, which conforms to all of our assumptions, we are interested in the probability distribution of all pages in the web after a random surfer chooses links an infinite number of times. This process is a Markov chain, and as a result, the final state does not depend on the initial condition, so we are free to allow our random surfer to start on the Algebra page. From there, the surfer has an equal probability of moving to the Abelian Group, Additive Inverse or Abstract Algebra page. Thus the probability to move to any page is $\frac{1}{3}$. Imagine the surfer moves to the Abelian Group page. The Abelian Group page only links to Abstract Algebra so the random surfer has a probability 1 of moving to the Abstract Algebra page. We can formulate a mathematical expression for the web of figure 1.1 as a 4×4 matrix, P. Every column, i, of P represents a single page, and every element of that column, p_{ij}, represents the probability a user on page i will randomly click the link leading to page j.

$$P = \begin{array}{c} \\ \mathit{AdditiveInverse} \\ \mathit{AbstractAlgebra} \\ \mathit{AbelianGroup} \\ \mathit{Algebra} \end{array} \begin{array}{c} \begin{array}{cccc} \mathit{AdditiveInverse} & \mathit{AbstractAlgebra} & \mathit{AbelianGroup} & \mathit{Algebra} \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & \frac{1}{3} \\ 0 & 0 & 1 & \frac{1}{3} \\ 1 & \frac{1}{2} & 0 & \frac{1}{3} \\ 0 & \frac{1}{2} & 0 & 0 \end{pmatrix} \end{array}$$

With all of our assumptions in place, the sum of every element in a column is $\sum_{j=1}^{4} p_{ij} = 1$ and every element is $0 \leq p_{ij} \leq 1$: stated physically, the probability of leaving a page is 1 and the probability of clicking any link is positive and less than or equal to 1. Therefore, P is a regular transition matrix with properties we can exploit to determine the probability distribution of the pages. If a random surfer clicks an infinite number of links, how many times did the random surfer enter each page compared to the total number of pages? If we randomly select an iteration, the larger the probability that the surfer was on a given page, the larger the rank of that page in the web. Let us now generalize our 4×4 matrix to a web of n pages.

Consider a $n \times n$ matrix, P, where $n =$ the number of pages. Each time the user clicks a link, we multiply the preceding transition matrix by P. As $k \to \infty, P^k$ approaches a matrix M with identical columns. A column of M is called the stability distribution, $\vec{\eta^*}$, because it represents the probability distribution of the pages when $P^k \approx P^{k+1}$ i.e. the iterative method has stabilized. The ith entry of the stability distribution is the rank of the ith page.

We will consider three computational methods of increasing analytic complexity but decreasing computational strain to determine the stability distribution. In the first two, we

assume no unlinked pages, sinks, or 'teleporting' surfers such that the Markov transition matrix is regular, and in the third, we introduce a damping factor that allows for unlinked pages, sinks and 'teleporting' surfers; however, we still assume random surfers. Method one computes the stability distribution by directly iterating the transition matrix P, while method two computes the stability distribution using eigenvalues and eigenvectors of P to reduce the computational load. Method three assumes only random surfers and restructures the transition matrix P to guarantee it is regular, then uses the Banach Fixed Point Theorem to generate a recursive linear operation that approaches the stability distribution as the number of recursions approaches infinity.

We created 3 webs of $4, 42$, and 299 pages respectively using the python wikipedia API. We fed the transition matrices representing these webs into the code attached in Appendix A and compared the time required to compute the stability distribution of each matrix with each method.

4 Mathematical Formulation

The first method is a brute force method that iterates the transition matrix, P, to compute the stability distribution. Because our assumptions guarantee P is regular, the Power Method Convergence Theorem (PMCT) guarantees that the series $\vec{z}, P\vec{z}, P^2\vec{z}, \dots P^k\vec{z}$ approaches $\vec{\eta^*}$ as $k \to \infty$ where $\vec{z} = (\frac{1}{n}, \frac{1}{n}, \dots, \frac{1}{n},)^t$ and $n =$ the number of pages.[1] This method is performed on three example webs in section 5. While this method is the easiest to analytically derive, it is unfeasible to compute as the number of pages increase to real world values. Thus, we can exploit eigenvalues and properties of regular transition matrices to ease the computational requirements.

The second method, relies on the Frobenius Theorem. Because of our assumptions, P is a regular transition matrix for which the Frobenius Theorem guarantees P possesses exactly one eigenvalue $\lambda^* = 1$ and an associated unique eigenvector $\vec{\eta^*}$ which is the stability distribution.1 Thus we can compute the stability distribution by solving the equation $(P - \lambda^* I)\vec{\eta^*} = \vec{0}$ via Gaussian Elimination. While this method is computationally easier than the first, it still becomes unfeasible if the number of pages approaches the billions considered in the real world, and it relies on untenable assumptions. This motivates us to search for a practical method that can handle billions of pages with more reasonable assumptions.

The third mmethod allows for an irregular Markov transition matrix i.e. sinks and 'teleporting' surfers, by introducing a damping factor β. Instead of performing computations on P, this method uses the modified matrix $P_\beta = (1-\beta)P + \beta Q$ where $\beta = [0,1]$ and Q is a matrix whose entries are all $q_{i,j} = \frac{1}{n}$ for all i and j where $n =$ the number of pages. β has historically been set to many values, and we will let $\beta = 0.3$, because this value was recently used by Google[2]. Even if P has zero-columns or rows, this modification fills those in such that P_β is always a regular transition matrix. The value of β can also be tweaked to account for how often surfers 'teleport', although we will not be concerned with this feature. Now that our modified transition matrix is guaranteed to be regular, we can calculate the stability distribution with the Banach Fixed Point Theorem (BFPT)[3]. This theorem states for a $n \times n$ regular transition matrix, P_β, there is a vector space $\mathbb{R}^n = \{\chi | \chi = (p_1, ...p_n)^t, p_i \in [0,1], \sum_{i=1}^n p_i = 1$ and a linear operator $L : \mathbb{R}^n \longrightarrow \mathbb{R}^n$ defined by $L[\chi] = P_\beta \chi$. To use this theorem we must show the linear operator L is a contraction on $\mathbb{R}^n$.

For $\vec{x} = \sum_{j=1}^n x_i \eta_i \in \vec{y} \sum_{j=1}^n y_i \eta_i \in \mathbb{R}^n$ where $\vec{\eta} = \{\eta_1, \ldots, \eta_n\}$ is an eigenvector basis of P_β and $q \in [0,1)$ and distance defined as $d(\vec{x}, \vec{y}) = \sum_{i=1}^n |x_i - y_i|$. If $d(L[\vec{x}], L[\vec{y}]) = d(P_\beta \vec{x}, P_\beta \vec{y}) \leq q \cdot d(\vec{x}, \vec{y})$, then L is a contraction on $\mathbb{R}^n$. We can express $P_\beta \vec{x}$ as $\begin{pmatrix} \sum_{j=1}^n p_{1j} x_j \\ \vdots \\ \sum_{j=1}^n p_{nj} x_j \end{pmatrix}$.

Then

$$d(P_\beta \vec{x}, P_\beta \vec{y}) = \sum_{i=1}^n |\sum_{j=1}^n p_{ij} x_j - \sum_{j=1}^n p_{ij} y_j|$$

Because $0 \leq p_{ij} \leq 1$ for all i and j, $\sum_{j=1}^n p_{ij} xj \leq x_i$ and $\sum_{j=1}^n p_{ij} yj \leq y_i$, thus

$$\sum_{i=1}^n |\sum_{j=1}^n p_{ij} x_j - \sum_{j=1}^n p_{ij} y_j| < q \cdot \sum_{i=1}^n |x_i - y_i|.$$

Therefore, L is a contraction on $\mathbb{R}^n$ and the BFPT guarantees a $\vec{\eta^*}$ such that $L[\vec{\eta^*}] = \vec{\eta^*}$ and moreover given any $\chi_0 \in \mathbb{R}^n$ we can recursively define $\chi_{k+1} = L[\chi_k]$ and $\vec{\eta^*} = \frac{lim}{k \to \infty} \chi_k$. Letting $\vec{\chi_0} = (\frac{1}{n}, \cdots, \frac{1}{n})^t$ our recursion algorithm becomes

$$\begin{aligned} \vec{\chi}_{k+1} &= L[\vec{\chi_k}] = P_\beta \vec{\chi_k} \\ &= (1-\beta) P \vec{\chi_k} + \beta Q \vec{\chi_k} \\ &= (1-\beta) P \vec{\chi_k} + \beta \vec{\chi_0}. \end{aligned}$$

Note that because the sum of the elements of $\vec{\chi_k}$ is 1 no matter the iteration, multiplication by Q always results in $\vec{\chi_0}$.

This method automatically handles unlinked pages, however, sinks still present a problem because a zero column in P will generally not be converted into a column of P_β whose elements sum to 1. However, because the damping factor allows for 'teleporting' surfers, a user who encounters a sink will be assumed to manually enter a random URL to another page. Thus if the ith page is a sink, it will be represented in P by $(\frac{1}{n-1}, \frac{1}{n-1}, \ldots, 0, \ldots, \frac{1}{n-1}, \frac{1}{n-1})^t$ where $n =$ the number of pages in the web, and the 0 element appears in the ith entry.

5 Numerical Examples

To show the effectiveness of Google's page rank algorithm, we created code that implemented the algorithm using Wikipedia page links. Wikipedia pages link to other Wikipedia pages which makes it a great candidate for the page rank algorithm. Using Linear algebra as the base page, we created a web using Linear algebra's page links with size 6,50, and 300 and ranked the page importance of in each case. The first part of our code is filtering the data. For our implementation of page rank the graph must be connected, meaning, any page that does not link to any other pages is thrown out. After filtering the data we are left with a matrices of size 4,42, 299 and 7, 262, and 8936 edges respectively. The code then creates a Markov transition matrix and uses the PMCT (method one), Gaussian Elimination (method two), and BFPT (method three) to determine the rank of each page. To illustrate this clearly, we will walk through the code using the web of size 4. The web of size 4 is shown in figure 1.1, from there the function *create_matrix()* will create the Markov transition matrix by finding all the out going edges of a link and then places the $\frac{1}{n}$ where n equals the total outgoing edges into the position of the outgoing link row. The web size 4 creates the following Markov transition matrix:

$$P = \begin{array}{c} \\ \mathit{AdditiveInverse} \\ \mathit{AbstractAlgebra} \\ \mathit{AbelianGroup} \\ \mathit{Algebra} \end{array} \begin{array}{c} \begin{array}{cccc} \mathit{AdditiveInverse} & \mathit{AbstractAlgebra} & \mathit{AbelianGroup} & \mathit{Algebra} \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & \frac{1}{3} \\ 0 & 0 & 1 & \frac{1}{3} \\ 1 & \frac{1}{2} & 0 & \frac{1}{3} \\ 0 & \frac{1}{2} & 0 & 0 \end{pmatrix} \end{array}$$

After creating the matrix then code will then find the eigenvector associated with eigenvalue of 1 using PMCT, Gaussian Elimination, and BFPT. Using sixty iterations for both the PMCT and BFPT gives us the following results.

$$\vec{\eta^*} = \begin{matrix} \\ \\ \\ \\ \end{matrix} \begin{matrix} PMCT \\ \begin{pmatrix} .06666624 \\ .3999821 \\ .33333448 \\ .20000106 \end{pmatrix} \end{matrix} \begin{matrix} GaussianElimination \\ \begin{pmatrix} .06666667 \\ .4 \\ .33333333 \\ .2 \end{pmatrix} \end{matrix} \begin{matrix} BFPT \\ \begin{pmatrix} .06666717 \\ .40000213 \\ .33333197 \\ .19999873 \end{pmatrix} \end{matrix}$$

We find that with 60 iterations PMCT and BFPT have five units of precision and that the page ranking is Abstract Algebra, Abelian Group, Algebra, and finally, Additive Inverse at the bottom. We then timed the speed of each method and with web of size 6,50,and 300 and found the following results in seconds:

$$Time = \begin{matrix} \\ Web_6 \\ Web_{50} \\ Web_{300} \end{matrix} \begin{matrix} PMCT \\ \begin{pmatrix} .000951s \\ .001901s \\ .173987s \end{pmatrix} \end{matrix} \begin{matrix} GaussianElimination \\ \begin{pmatrix} .016232s \\ .0267722s \\ .088579s \end{pmatrix} \end{matrix} \begin{matrix} BFPT \\ \begin{pmatrix} .001188s \\ .001011s \\ .001219s \end{pmatrix} \end{matrix}$$

Showing that as we approach infinity then BFPT is by far the least computational expensive followed by Gaussian Elimination, and finally PMCT.

6 Discussion and Conclusions

We all learned a lot of new information while we were researching this topic. None of us knew much about Markov chains or the math behind search engines when we started. We ultimately found multiple ways to calculate the importance of a set of links and how it connects to linear algebra. This information we found wasn't exactly what we expected. While we all assumed a system as complicated as PageRank would have several advanced calculations, none of us expected it to use linear algebra to the extent it did. We were surprised how many methods could produce a stability distribution from elementary Gaussian Elimination to a recursive method based on BFPT.

An interesting result of our computations was that for small webs, PMCT was faster than Gaussian Elimination was faster than BFPT. However, as the web size increased from 4 to 262 pages, this trend reversed. PMCT became dramatically worse: increasing by 0.173 seconds, while Gaussian Elimination increased by 0.072 seconds. Surprisingly, increasing

web size had little impact on BFPT, the 262-web only required 0.00003 more seconds to compute than the 4-web and in fact, the 42-web was calculated faster than the 4-web. While we expected the BFPT method to be the fastest, we did not expect increasing the web size to have such a negligible impact on computation time. An interesting follow-up would be to examine the relationship between computation time for all three methods and the size of the web.

Although our computations are specific to PageRank, the math behind them can be generalized. Markov chains are used for a wide variety of math, including economics, transportation, and population models. However, our more specific findings could be applied to systems similar to PageRank, where the references to other items is used to rank their relative importance. For example, a database could be created that ranks scholarly papers and articles. The references and citations could be analyzed in the same way as was done for the links of webpages by PageRank, and a ranking could be established of the relative importance. All possible applications tie back to our initial question, being how Google PageRank evaluates the importance of a variety of interconnected nodes. But for any future applications, there is still a great deal of research to be done. Our report only looked into the first version of one service offered by one of the several search engines on the internet. More recent versions of PageRank, other Google Search Engines such as Google Maps or Google Images, and other search engines such as Bing and Yahoo could also be researched. Seeing as this report found multiple methods to calculate the relative importance from the transition matrix, there's no telling what these other algorithms could offer - if not an entirely different model.

7 References

[1]Tanase R, Radu R. The Mathematics of Web Search. Ithaca, New York: Cornell Edu; 2009. Available from:
http://pi.math.cornell.edu/ mec/Winter2009/RalucaRemus/Lecture3/lecture3.html

[2]Bryan K, Leise T. The $25,000,000,000 Eigenvector: The Linear Algebra behind Google. SIAM REVIEW Vol. 48, No. 3, pp. 569–581. Available from:
https://epubs.siam.org/doi/pdf/10.1137/050623280

[3]Rousseau C.Klein Project. 2015. Available from:
http://blog.kleinproject.org/?p=280