Application of Linear Algebra in Search Engine Algorithms

Index:

Introduction:

In the earliest versions of search engines the occurrence of the keyword was the only important rule to decide if a document is relevant or not so naturally the document with the highest number of occurrences of keywords received the highest score based on the traditional text retrieval model.This method has its own limitations like having to do too much of the text processing redundantly and to retrieve the data in this was very time consuming and mainly searching this way would retrieve too many pages and many of them were not related to the search and so it was not very user friendly.

Most recent search engines try to overcome this drawback as a non user friendly search engine by using a two-step process to retrieve pages related to a user's query. In the first step, traditional text processing is done to find all documents using the query terms, or related to the query terms by semantic meaning. With the massive size of the web, this first step can result in thousands of retrieved pages related to the query. To make this list manageable for a user, many search engines sort this list by some ranking criteria .

One popular way to create this ranking is to exploit the additional information inherent in the web due to its hyperlinking structure. Link analysis algorithms for Web search engines are used in such exploitation , which determine the importance and relevance of Web pages. Among the link analysis algorithms, PageRank is the state of the art ranking mechanism that is used in Google today. This algorithm is modeled as the behavior of a randomized Web surfer; this model can be seen as a Markov chain to predict the behavior of a system that travels from one state to another state considering only the current condition.

More about Markov Chain and theory:

Most of our study of probability has dealt with independent trials processes. These processes are the basis of classical probability theory and much of statistics. Modern probability theory studies chance processes for which the knowledge of previous outcomes influences predictions for future experiments. In principle, when we observe a sequence of chance experiments,

all of the past outcomes could influence our predictions for the next experiment.

A.A.Markov began the study of a new chance process , in which the outcome of a given experiment can affect the outcome of the next experiment .It is a “memoryless” property. In other words , the next state of the process only depends on the previous state and not the sequence of states.

With this theory of Markov , Larry Page and Sergey Brin developed Google’s PageRank algorithm in 1998, still during the early stages of the Internet. By 2002, 2 years before Google’s IPO, PageRank was operating on a matrix at the order of 2.7 billion, cited by Cleve Moler as ”The World’s Largest Matrix Computation” . Today Google searches over 30 trillion webpages, 100 billion times a month, and those numbers are still growing as new sites are used and new users Google on a daily basis. The algorithm, that was far more efficient than its rival ’90s search engines at ranking web pages based on a given search, is still exceedingly relevant today, with some minor modifications over the years. To understand Google’s PageRank is to understand the development and structure of the Internet. The goal of this report is to highlight what makes the Google PageRank algorithm significantly superior to its rival search engines by explaining its core mathematics.

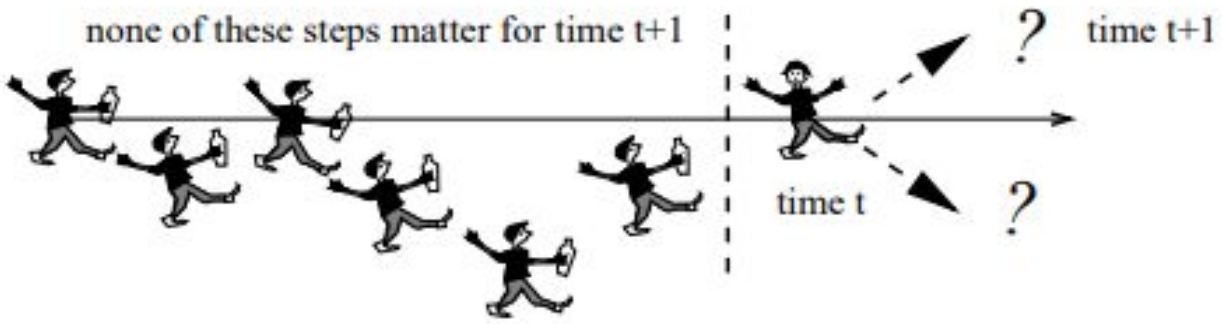
Report:

Markov Chains

Markov chains are used to compute the probabilities of events occurring by viewing them as states transitioning into other states, or transitioning into the same state as before. The following lines help us in understanding this concept better.

We have a set of states, S={s1,s2.... s_n } .The process starts in one of these states and moves successively from one state to another. If the chain is currently in state s_i , then it moves to state s_j at next step with a probability p_{ij} , and this probability does not depend on which states the chain has previously been in .

The below image is a good example of this concept. This is an image of a drunk



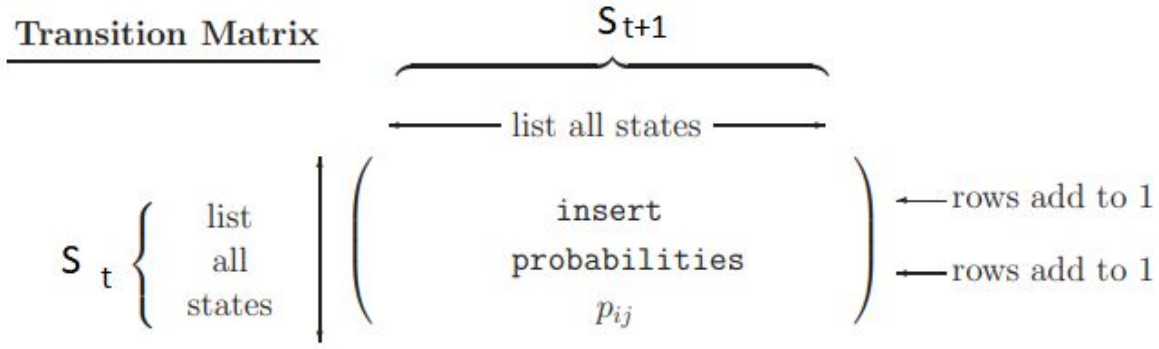
man walking on the street. His next step has nothing to do with the past and also it only depends on the present.

Some important definitions in this concept:

- The **state** of a Markov chain at time t is the **value of** s_t .
- The **state space** of a Markov chain **s** , is the set of values that each s_t can take.
- A **trajectory** of a Markov chain is a **particular set of values for s1,s2....** s_t **.**

Markov Matrix

The matrix describing the Markov chain is called the transition matrix. It is the most important tool for analysing Markov chains.



Typically , a Markov matrix's entries represent transition probabilities from one state to another. The above structure represents the Markov matrix.Where s_t is the current state and s_{t+1} is the next state . It is a square matrix whose columns are probability vectors where,

- The **rows** represent **now**, or **from** s_t .
- The **columns** represent **next**, or **to** s_{t+1} .
- Entry **(i, j)** is the **conditional probability** that **next = j**, given that **now** = i: the probability of going **from** state **i** to state **j**.
  So the transition probabilities of a markov chain are given as:

$$p_{ij} = \mathrm{P}(s_{t+1} = \mathrm{j} \mid s_t = \mathrm{i}).$$

And also,

- Each column's entries sum up to 1 .
- Does not contain any negative entries .
- The row entries do not necessarily sum to 1.

Eigenvalues and Eigenvectors

In linear algebra, an **eigenvector** or **characteristic vector** of a linear transformation is a nonzero vector that changes at most by a scalar factor when that linear transformation is applied to it. The corresponding **eigenvalue** is the factor by which the eigenvector is scaled.

Geometrically, an eigenvector, corresponding to a real nonzero eigenvalue, points in a direction in which it is stretched by the transformation and the eigenvalue is the factor by which it is stretched. If the eigenvalue is negative, the direction is reversed.Loosely speaking, in a multidimensional vector space, the eigenvector is not rotated. However, in a one-dimensional vector space, the concept of rotation is meaningless.

That symbolically means,

$$Av = \lambda v,$$

Where A is an *n* by *n* matrix, and λ is a scalar variable and v is a vector.

Eigenvectors and Eigenvalues in Matrices

Linear transformations over a finite-dimensional vector space can be represented using matrices,which is especially common in numerical and computational applications.

Consider *n*-dimensional vectors that are formed as a list of *n* scalars, such as the three-dimensional vectorsThese vectors are said to be scalar multiples of each other, or parallel or collinear, if there is a scalar λ such that

$$x = \lambda y.$$

Now consider the linear transformation of *n*-dimensional vectors defined by an *n* by *n* matrix *A*,

$$Av = w,$$

Or

$$\begin{bmatrix} A_{11} & A_{12} & \dots & A_{1n} \\ A_{21} & A_{22} & \dots & A_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{n1} & A_{n2} & \dots & A_{nn} \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}$$

where, for each row,

$$w_i = A_{i1}v_1 + A_{i2}v_2 + \cdots + A_{in}v_n = \sum_{j=1}^{n} A_{ij}v_j$$

If it occurs that *v* and *w* are scalar multiples, that is if

$$Av = w = \lambda v,$$

then *v* is an **eigenvector** of the linear transformation *A* and the scale factor λ is the **eigenvalue** corresponding to that eigenvector. And the following vector is the **eigenvalue equation** for the matrix *A*.

$$(A - \lambda I)v = 0,_{3w}$$

where I is the n by n identity matrix and 0 is the zero vector.

Power Method for Converging on Eigenvector

When the equation for solving the eigenvector is small we could use simple linear algebra concepts like solving a linear or a quadratic equation to get the job done , but as the size of the equation increases we need many more complex methods which can approximate and ignore the redundant variables, compute and get the final results. One such algorithm is the iterative power method.

In mathematics, power iteration (also known as the power method) is an eigenvalue algorithm: given a diagonalizable matrix ***A***, the algorithm will produce a number λ , which is the greatest eigenvalue of ***A***, and a nonzero vector ***v***, which is a corresponding eigenvector of λ , that is, **Av=** λ **v.**

Power iteration is a very simple algorithm, but it may converge slowly. The most time-consuming operation of the algorithm is the multiplication of matrix ***A*** by a vector, so it is effective for a very large sparse matrix with appropriate implementation. But, the method can be used only to find the eigenvalue of A that is largest in absolute value—we call this eigenvalue the dominant eigenvalue of A. Although this restriction may seem severe, dominant eigenvalues are of primary interest in many physical applications.

The power method for approximating eigenvalues is iterative. First we assume that the matrix ***A*** has a dominant eigenvalue with corresponding dominant eigenvectors. Then we choose an initial approximation of one of the dominant eigenvectors of ***A***. This initial approximation must be a nonzero vector in R^n . Finally we form the sequence given by

$$x_1 = A\,x_0$$

$$x_2 = A\,x_1 = A(A\,x_0) = A^2 x_0$$

$$x_3 = A\,x_2 = A(A^2\,x_0) = A^3 x_0$$

$$\vdots$$

$$x_k = A\,x_{k-1} = A(A^{k-1}\,x_0) = A^k x_0$$

For large powers of k, and by properly scaling this sequence, we will see that we obtain a good approximation of the dominant eigenvector of ***A***.

Rayleigh quotient

The following theorem provides a formula for determining the eigenvalue corresponding to a given eigenvector. This theorem is credited to the English physicist John William Rayleigh (1842–1919).

If x is an eigenvector of a matrix ***A***, then its corresponding eigenvalue is given by.

$$\lambda = \frac{Ax \,.\, x}{x.x}$$

Where x is the eigenvector of ***A*** and this quotient is called the Rayleigh quotient.

Perron-Frobenius theorem

For regular matrices

Suppose $\boldsymbol{A} \in R^{n\times n}$ is nonnegative and regular, i.e., $A^k > 0$ for some k then

- There is an eigenvalue λ_{pf} of ***A*** that is real and positive, with positive left and right eigenvectors .
- For any other eigenvalue λ , we have $|\lambda| < \lambda_{pf}$.

- The eigenvalue λ_{pf} is simple, i.e., has multiplicity one, and corresponds to a 1×1 Jordan block .

The eigenvalue λ_{pf} is called the Perron-Frobenius (PF) eigenvalue of ***A.***
The associated positive (left and right) eigenvectors are called the (left and right) PF eigenvectors (and are unique, up to positive scaling).

Definition*:*
Jordan block: A ***Jordan block*** *is a square* ***matrix*** *which has zero entries everywhere except on the diagonal, where the entries are a fixed scalar, and except on the superdiagonal, where the entries are either all 0s or all 1s.*

For nonnegative matrices

Suppose $\boldsymbol{A} \in R^{n \times n}$ and $A \geq 0$, then

- There is an eigenvalue λ_{pf} of A that is real and nonnegative, with associated nonnegative left and right eigenvectors.
- For any other eigenvalue λ of ***A***, we have $|\lambda| \leq \lambda_{pf}$

λ_{pf} is called the Perron-Frobenius (PF) eigenvalue of ***A.***

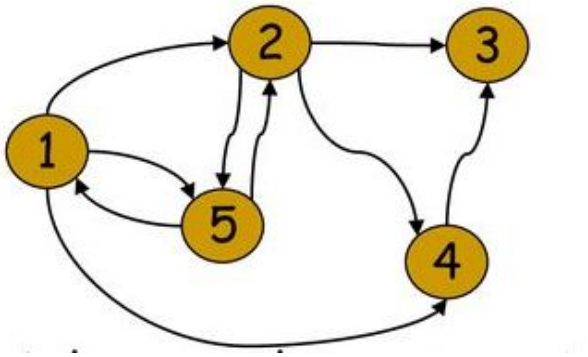
The associated nonnegative (left and right) eigenvectors are called (left and right) PF eigenvectors in this case, they need not be unique, or positive.

Markov model of the web

We can represent the hyper linked structure of the web in the form of a directed graph. Where, if a web page is connecting to another web page then we add these webpages to the graph and make a directional link between them.

And later we represent the same graph with a square matrix P whose each element p_{ij} represents the probability of moving from page i to page j in one time step.

These probabilities can be calculated using the following methods.



$$P = \begin{pmatrix} 0 & 1/3 & 0 & 1/3 & 1/3 \\ 0 & 0 & 1/3 & 1/3 & 1/3 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1/2 & 1/2 & 0 & 0 & 0 \end{pmatrix}$$

Page Rank

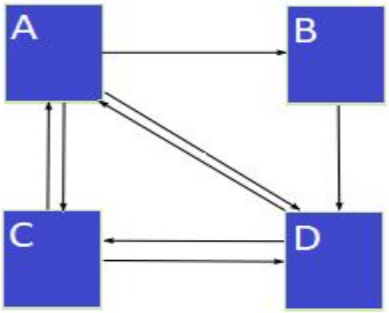
A PageRank is a calculation which evaluates the quality and quantity of links to a webpage to determine a relative score of that page's importance and authority . The underlying assumption is that more important websites are likely to receive more links from other websites .

The original Brin and Page model for PageRank uses the hyperlink structure of the web to build a Markov chain with a primitive transition probability matrix P.The irreducibility of the chain guarantees that the long-run stationary vector πT , known as the PageRank vector, exists. It is well-known that the power method applied to a primitive matrix will converge to this stationary vector . Further, the convergence rate of the power method is determined by the magnitude of the subdominant eigenvalue of the transition rate matrix.

Algorithm

We are going to explain the page rank algorithm using the following example.

Ex: *There are four pages. Page A contains a link to page B, a link to page C, and a link to page D. Page B contains one single link to page D. Page C points to pages A and D, and page D points to pages A and C. They are represented by the following graph. We have L(A) = 3, L(B) = 1 and L(C) = L(D) = 2(where L(p) be the number of outgoing links in a page p.)*



Let N be the total number of pages. We create an N × N matrix ***A*** by defining the *(i, j)* entry as

$$p_{ij} = \left\{ \frac{1}{L(i)} \quad if\ there\ is\ a\ link\ from\ j\ to\ i \ , \ \ 0 \ \ otherwise \right\}$$

Here the matrix ***A*** is 4 × 4 matrix

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

The simplified Pagerank algorithm is:

Initialize x to an N ×1 column vector with non-negative components, and then repeatedly replace x by the product Ax until it converges.

We call the vector x the ***pagerank vector***. Usually, we initialize it to a column vector whose components are equal to each other.

We can imagine a bored surfer who clicks the links in a random manner. If there are k links in the page, he/she simply picks one of the links randomly and goes to the selected page. After a sufficiently long time, the N components of the pagerank vector are directly proportional to the number of times this surfer visits the N web pages.

For example , in the above example we let the components of the vector x be xA, xB, xC and xD. Initialize x to be the all-one column vector, i.e.,

$$\mathbf{x} = \begin{bmatrix} x_A \\ x_B \\ x_C \\ x_D \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}.$$

The evolution of the pagerank vector is shown in the following table.

iteration	x_A	x_B	x_C	x_D
0	1	1	1	1
1	1	0.3333	0.8333	1.8333
2	1.3333	0.3333	1.25	1.0833
3	1.667	.4444	.9861	1.4028
4	1.1944	0.3889	1.0903	1.3264
5	1.2083	0.3981	1.0613	1.3322
6	1.1968	0.4028	1.0689	1.3316
7	1.2002	0.3989	1.0647	1.3361

We observe that the algorithm converges quickly in this example.Within 10 iterations, we can see that page D has the highest rank. In fact, page D has 3 incoming links, while the others have either 1 or 2 incoming links. It conforms with the rationale of the pagerank algorithm that a page with larger incoming links has higher importance.

How to handle dangling nodes ?

A node is called a dangling node if it does not contain any out-going link, i.e., if the out-degree is zero.

The simplified Pagerank algorithm collapses if there is a dangling node in the web graph because if we initialize the vector x to the all-one vector, the simplified Pagerank algorithm gives values such that the pagerank vector will converge to zero ultimately.

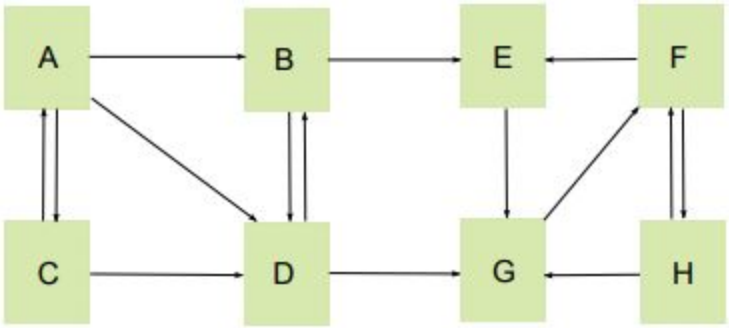
One remedy is to modify the simplified Pagerank algorithm by replacing any all-zero column by

$$\begin{bmatrix} 1/N \\ 1/N \\ \vdots \\ 1/N \end{bmatrix}$$

This adjustment is justified by modeling the behaviour of a web surfer, who after reading a page with no out-going link, he/she will jump to a random page. He/she simply picks one of the N pages randomly with equal probability. This model may not be realistic, but it simplifies the computation of the algorithm.

How to handle reducible webgraph ?

Consider the following web graph consisting of eight web pages.



There is no dangling node. But, once a surfer arrives at pages E, F, G or H, he/she will get stuck in these four pages. The matrix A of this web graph is

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

If we initialize the pagerank vector x to be the 8 × 1 column vector with components all equal to 1/8, then the Pagerank algorithm gives

iteration	x_A	x_B	x_C	x_D	x_E	x_F	x_G	x_H
0	0.125	0.125	0.125	0.125	0.125	0.125	0.125	0.125
1	0.0625	0.1042	0.0417	0.1667	0.125	0.1875	0.25	0.0625
2	0.0208	0.1042	0.0208	0.0937	0.1458	0.2813	0.2396	0.0938
3	0.0104	0.0538	0.0069	0.0697	0.1927	0.2865	0.2396	0.1406
4	0.0035	0.0382	0.0035	0.0339	0.1701	0.3099	0.2977	0.1432
5	0.0017	0.0181	0.0012	0.0220	0.1740	0.3694	0.2587	0.1549
6	0.0006	0.0116	0.0006	0.0102	0.1937	0.3362	0.2625	0.1847
7	0.0003	0.0053	0.0002	0.0063	0.1739	0.3549	0.2912	0.1681
8	0.0001	0.0032	0.0001	0.0028	0.1801	0.3752	0.2610	0.1774

We can see that the ranking of pages A to D drops to zero eventually. But page D has three incoming links and should have some nonzero importance. The Pagerank algorithm does not work in this example.

This pathological web graph belongs to the category of ***reducible*** graph. In general, a graph is called ***irreducible*** if for any pair of distinct nodes, we can start from one of them, follow the links in the web graph and arrive at the other node, and vice versa. A graph which is not irreducible is called reducible. In this example, there is no path from E to B, and no path from G to D. The graph is therefore reducible. In order to calculate page ranks properly for a reducible web graph, Page and Brin proposed that take a weighted average of the matrix $\bar{A}$ with an all-one N × N matrix. Define the matrix.

$$\mathbf{M} = d\bar{\mathbf{A}} + \frac{1-d}{N}\begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & 1 & 1 & \dots & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & 1 & \dots & 1 \end{bmatrix}$$

where $\bar{A}$ is the modified matrix defined in the previous section, and d is a number between 0 and 1. The constant d is usually called the damping factor or the ***damping constant***. The default value of ***d is 0.85*** in Google.

With d set to 0.85, the matrix M for the web graph in the above is

iteration	x_A	x_B	x_C	x_D	x_E	x_F	x_G	x_H
0	0.125	0.125	0.125	0.125	0.125	0.125	0.125	0.125
1	0.0719	0.1073	0.0542	0.1604	0.1250	0.1781	0.2313	0.0719
2	0.0418	0.1073	0.0391	0.1077	0.1401	0.2459	0.2237	0.0945
3	0.0354	0.0764	0.0306	0.0928	0.1688	0.2491	0.2237	0.1232
4	0.0317	0.0682	0.0288	0.0742	0.1571	0.2613	0.2541	0.1246
5	0.0310	0.0593	0.0277	0.0690	0.1588	0.2877	0.2368	0.1298
6	0.0305	0.0568	0.0275	0.0645	0.1662	0.2752	0.2382	0.1410
7	0.0304	0.0548	0.0274	0.0633	0.1598	0.2811	0.2474	0.1357
8	0.0304	0.0543	0.0274	0.0623	0.1615	0.2867	0.2392	0.1382

Using this matrix in each iteration, the evolution of the pagerank vector is

iteration	x_A	x_B	x_C	x_D	x_E	x_F	x_G	x_H
0	0.125	0.125	0.125	0.125	0.125	0.125	0.125	0.125
1	0.0719	0.1073	0.0542	0.1604	0.1250	0.1781	0.2313	0.0719
2	0.0418	0.1073	0.0391	0.1077	0.1401	0.2459	0.2237	0.0945
3	0.0354	0.0764	0.0306	0.0928	0.1688	0.2491	0.2237	0.1232
4	0.0317	0.0682	0.0288	0.0742	0.1571	0.2613	0.2541	0.1246
5	0.0310	0.0593	0.0277	0.0690	0.1588	0.2877	0.2368	0.1298
6	0.0305	0.0568	0.0275	0.0645	0.1662	0.2752	0.2382	0.1410
7	0.0304	0.0548	0.0274	0.0633	0.1598	0.2811	0.2474	0.1357
8	0.0304	0.0543	0.0274	0.0623	0.1615	0.2867	0.2392	0.1382

We see that the page ranks of pages A to D do not vanish.

Updating the PageRank vector

The computation of the eigenvector used in PageRank is expensive. Hence, Google does not update it in line with the Web's updation. This also leads to restarting of computations.

There is research focussing on techniques to reuse previous computations to obtain a new PageRank vector. Some of these are:

1. Monte carlo methods and asymptotic analysis
2. Sequential updating

Probabilistic Monte Carlo methods have several advantages over deterministic power iteration method:

1. Provide good estimation of the PageRank for relatively important pages after a single iteration.
2. Have natural parallel implementation for increased computation speedup.
3. Allow continuous update of the PageRank vector as the Web structure updates.

HITS Algorithm(Hyperlink Induced Topic Search)

In the same time that PageRank was being developed, Jon Kleinberg a professor in the Department of Computer Science at Cornell came up with his own solution to the Web Search problem. He developed an algorithm that made

use of the link structure of the web in order to discover and rank pages relevant for a particular topic.

HITS increases the chances of finding web pages that are relevant, authoritative i.e. contain usable information about the query. Hubs aggregate all authoritative links.

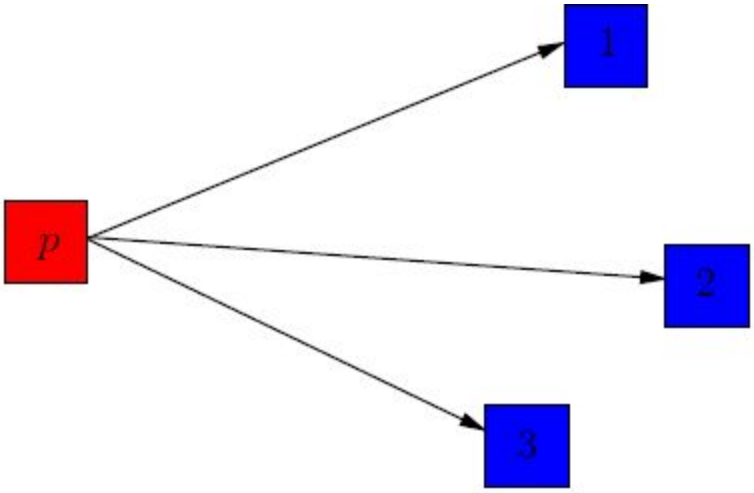
HITS identifies good authorities and hubs for a topic by assigning two numbers to a page: an authority and a hub weight. These weights are defined recursively. Hubs and authorities exhibit a *mutually reinforcing relationship.*A higher authority weight occurs if the page is pointed to by pages with high hub weights. A higher hub weight occurs if the page points to many pages with high authority weights.

Consider a query Q. First collect the top 200 documents that contain the highest number of occurrences of the search phrase Q. This set is called Root(R_Q). It is a mostly disconnected graph with the nodes having very few, if any, links to each other. Hence, PageRank techniques cannot be applied onto R_Q.

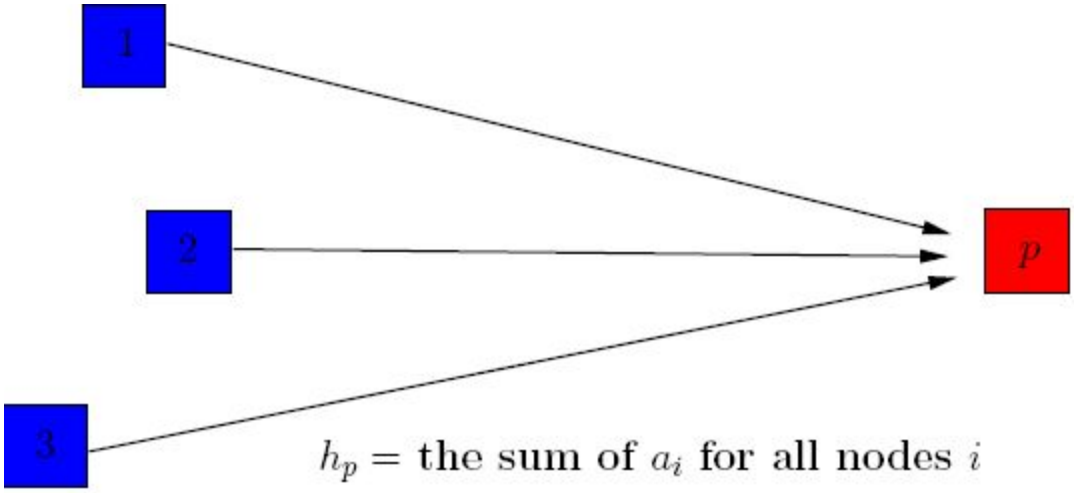
Authorities for the query Q are likely to be pointed out by at least one page in R_Q. So the subgraph R_Q is extended by including all edges coming from or pointing to nodes from R_Q. We denote by S_Q the resulting subgraph and call it the *seed* of our search. S_Q is a reasonably small graph, with many authoritative sources for A. Higher rated authorities will have a greater number of common pages compared to S_Q, with a great overlap of hubs.

Mathematically,

A page *i* has: an authority weight a_i, and a hub weight h_i. Pages with a higher a_i are better authorities, and pages with a higher h_i are better hubs.



a_p = the sum of h_i for all nodes i pointing to p



h_p = the sum of a_i for all nodes i pointed to by p

A hub is good if it increases the authority weight of the pages it points to. A good authority increases the hub weight of the pages that point to it. Apply the two operations alternatively till there is a tradeoff between the hub and authority weights and equilibrium is achieved.

Let A be the adjacency matrix of the graph S_Q and denote the authority weight vector by v and the hub weight vector by u, where

$$v = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix} \quad \text{and} \quad u = \begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \end{bmatrix}$$

Let us notice that the two update operations described in the pictures translate to:

$$\begin{cases} v = A^t \cdot u \\ u = A \cdot v \end{cases}$$

.

If we consider that the initial weights of the nodes are

$$u_0 = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} \quad \text{and} \quad v_0 = A^t \cdot \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}$$

then, after k steps we get the system:

$$\begin{cases} v_k = (A^t \cdot A) \cdot v_{k-1} \\ u_k = (A \cdot A^t) \cdot u_{k-1} \end{cases}$$

.

The HITS algorithm utilises the Perron-Frobenius theorem in the background. It can be summarised as:

1. If $v_1, \ldots, v_k, \ldots$ is the sequence of authority weights we have computed, then $\mathbf{V}_1, \ldots, \mathbf{V}_k, \ldots$ converges to the unique probabilistic vector corresponding to the dominant eigenvalue of the matrix A^tA. With a slight abuse of notation, we denote here by $\mathbf{V}_k$ the vector v_k normalized so that the sum of its entries is 1.
2. Likewise, if $u_1, \ldots, u_k, \ldots$ are the hub weights that we have iteratively computed, then $\mathbf{U}_1, \ldots, \mathbf{U}_k, \ldots$ converges to the unique probabilistic vector corresponding to the dominant eigenvalue of the matrix AA^t. We use the same notation, that $\mathbf{U}_k = (1/c)u_k$, where c is the scalar equal to the sum of the entries of the vector u_k.

HITS algorithm is in the same spirit as PageRank. They both make use of the link structure of the Web graph in order to decide the relevance of the pages.

The difference is that unlike the PageRank algorithm, HITS only operates on a small subgraph (the seed S_Q) from the web graph. This subgraph is query dependent; whenever we search with a different query phrase, the seed changes as well. HITS ranks the seed nodes according to their authority and hub weights. The highest ranking pages are displayed to the user by the query engine.

SALSA

SALSA is a new stochastic approach for link-structure analysis, which examines random walks on graphs derived from the link-structure. SALSA and Kleinberg's Mutual Reinforcement approach employ the same meta-algorithm. SALSA is equivalent to a weighted in degree analysis of the link-structure of WWW subgraphs, making it computationally more efficient than the Mutual reinforcement approach.

SALSA combines the random walk idea of PageRank with the hub/authority idea of HITS.

Given a graph G, a bipartite undirected graph H is constructed by building the subset V_a of all the nodes with positive indegree (the potential authorities), and the subset V_h of all the nodes with positive outdegree (the potential hubs). After renumbering, the elements of V_a and V_h become the nodes of H. Since some nodes may have both positive indegree and outdegree, the number m of nodes of H satisfies $m \leq 2n$, where n is the number of nodes of G. The (undirected) edges of H are defined from G as follows: if G has a link from i to j, then we put an edge between the nodes corresponding to i in V_h and j in V_a. The algorithm corresponds to a two-step random walk on the graph H.

If we pick a node in V_a and randomly follow an edge, we traverse to a node in V_h. A second step returns us to V_a. Thus a pair of two-step random walks, one starting in V_a and the other in V_h, can be used to determine the hub and authority vectors.

As in the PageRank scheme, each random walk is a Markov chain with a corresponding transition probability matrix. As before, let A be the adjacency matrix of G. Let Wr be the matrix generated from A by dividing each entry of A

by its row sum. Similarly define Wc to be generated by dividing each entry of A by its column sum. We arrive at $\vec{a}_k = W_c^T W_r \vec{a}_{k-1}$ and $\vec{h}_k = W_r W_c^T \vec{h}_{k-1}$. Note that the sequence $\vec{a}_k$ depends on the initialization $\vec{a}_0$, and $\vec{h}_k$ depends on the initialization $\vec{h}_0$.

This contrasts with HITS, where both sequences depend on $\vec{h}_0$, and leads to subtle differences in the behavior of the two algorithms. In particular the SALSA limits $\vec{a}$ and $\vec{h}$ may not satisfy $\vec{a} = W_c^T \vec{h}$.

Comparisons and Conclusions

These algorithms share a common underpinning; they find a dominant eigenvector of a nonnegative matrix that describes the link structure of the given network and use the entries of this eigenvector as the page weights.

However, PageRank and SALSA compute this eigenvector using a Markov chain method, while HITS is formulated in terms of summing the weights of linked nodes at each step.

Among the three, the weakest conclusions are for HITS and SALSA. Some of the reasons for this are the problem of repeated largest eigenvalue as the initialization changes and inappropriate zero weights. The premise of HITS and SALSA is that good hubs point to good authorities, if four out of five equally good hubs all point to one node and no others, then we expect that node to be a better authority than any of the four nodes pointed to by the fifth hub. In other words, we expect that $\vec{a} = W_c^T \vec{h}$ and $\vec{h} = W_r \vec{a}$. The fact that this is not the case results from the repeated eigenvalue of $W_c^T W_r$ .The output of graphs that give rise to repeated Eigenvalues is sensitive to the initial vector chosen.

The HITS algorithm with standard input yields different authority vectors for different initial seed vectors, and the SALSA algorithm with standard input gives inconsistent authority and hub vectors.

Literature Survey

1. https://www.wikipedia.org/
2. http://visual.icse.us.edu.pl/LA/problem_wlasny_Google.html
3. Applications of Matrix Computations to Search Engines An introduction to Markov chains
4. Anders Tolver. An Introduction to Markov Chains. University of Copenhagen. 2016
5. Amy N. Langville and Carl D. Meyer. Google's PageRank and Beyond: The Science of Search Engine Rankings. Princeton University Press. 2006
6. https://www.researchgate.net/figure/The-procedure-to-handle-dangling-nodes_fig1_321056905
7. (PDF) Updating PageRank with Iterative Aggregation - Amy Langville, Carl D. Meyer
8. Notes on PageRank Algorithm ,Lecturer: Kenneth Shum http://home.ie.cuhk.edu.hk/~wkshum/papers/pagerank.pdf
9. The Effect of New Links on Google PageRank - Nelly Litvak, Konstantin Avrachenkov
10. http://pi.math.cornell.edu/~mec/Winter2009/RalucaRemus/index.html
11. https://dl.acm.org/doi/abs/10.1145/382979.383041 - SALSA: the stochastic approach for link-structure analysis - R.Lempel, S. Moran
12. . S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. Computer networks and ISDN systems, 30(1-7):107–117, 1998
13. AUTHORITY RANKINGS FROM HITS, PAGERANK, AND SALSA: EXISTENCE, UNIQUENESS, AND EFFECT OF INITIALIZATION - Ayman Farahat, Thomas Lofaro, Joel C. Miller, Gregory Rae, Andlesley A. Ward
14. Python3 example: https://medium.com/analytics-vidhya/the-algorithm-behind-google-search-an-implementation-with-python-d6418023bbd9