Ukrainian Catholic University

Faculty of Applied Sciences

Business Analytics & Computer Science Programmes

The PageRank

Linear Algebra final project report

Authors:
Bohdan Ruban
Olexiy Hoyev
Ostap Trush

29 May 2022

Abstract

This work studies the Google's PageRank Algorithm[5] for web page ranking as well as different versions of it[2][3], which we implemented, like the Power method or the Eigendecomposition. We also compared out implementations to the already existed solutions and tested their effectiveness on the the real data.

1 Introduction

Ever since the World Wide Web was invented in 1989 by Tim Berners-Lee and the creation of the first search engine in 1990 there was a need to somehow manage web pages effectively. And after CERN placed the World Wide Web technology in the public domain in 1993 this problem became even bigger. How do you rank the importance of each web page in a huge cluster of them?

Google came up with their PageRank algorithm in 1998, which allowed them to skyrocket their search engine success, because of how effective the algorithm was in comparison to its predecessors like HITS. And even though this algorithm is more than 20 years old, in 2017 Google's Gary Illyes confirmed on Twitter that they were still using PageRank, just a "forged" variation of it.

In this report we try to experiment with different PageRank approaches and how effective they are on real data as well as compare our PageRank implementations with the ones available to us through professional third-party libraries.

2 Problem Setting

The entire Web, as it is, can be represented as a graph G, where all the n nodes of this graph are the web pages and every edge e between some two nodes is a directed link from one page to another or vise versa. The importance or relevance of each web page can then be evaluated based on a number of incoming links to it.

The "credibility" of those links can also be taken into consideration, which means that if a web page A is pointed to by a credible web page B that has a lot of incoming links, that link will be more important than, say a link from a web page C, which has 0 incoming links. This strength of a link is sometimes referred to as "link juice".

This terminology originated from PageRank which uses a very similar technique!

Getting the rank of each web page right is incredibly important, especially in 2022, when the first web page that is encountered is often the one being clicked, and it is essential that it contains useful and credible information so that the user does not waste time on information he/she does not need.

3 A short review of the related works and possible solution approaches

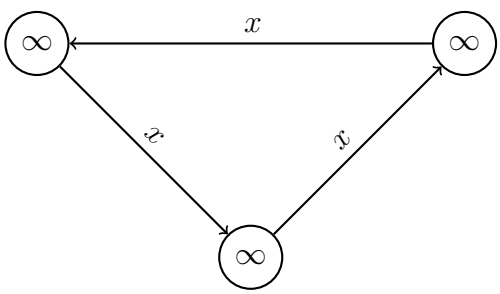
Even though Google's PageRank is the most popular page ranking algorithm by far, its definitely not the only one out there. OPIC, HITS and SALSA(which is considered more stable to "spammers" than HITS) are the other algorithms for ranking pages. In this section we will take a look at HITS as it is by far the most documented and effective page ranking method except PageRank, and compare it to different PageRank methods to see why the latter ended up winning the "ranking race".

To start with, Google's PageRank ranks an importance of a web page depending on how many links from other important web pages point to it. HITS or Hyperlink-Included Topic Search, however, divides the web page importance into two groups: Hubs and Authorities. Given a search query, Authorities are taken from a set of highly relevant web pages and are high-quality if many high quality pages link to them. Hubs are taken from a set of not very high relevant web pages, but are considered high quality if they link to many other relevant and high quality web pages. Then, each web page has an authority score which is the sum of the hub score of the web pages pointing to it, and a hub score, which is the sum of authority scores the web page is pointing to respectively.

Even though this seems as a better ranking method than PageRank, as it takes more inputs into consideration, it has some serious drawbacks. As a reader could have guessed from a description, HITS is query-dependent, which means that the search time for when a user enters a query into the search engine can be quite large because all those score calculations are done "on the fly". Moreover, HITS works on smaller graphs than PageRank, which can be seen as more effective and give better results and only the results, relevant to the query, but sometimes taking the whole picture into consideration can result in better ranking results for every user.

And that's where Google's PageRank takes an edge over HITS, because it is not query-dependent and already knows what rank each web page has, even before the user types in the query. Moreover PageRank can be personalized from user to user, still giving a better personalized ranking that HITS. And what is more important, is because it takes the whole Web graph into consideration, it is more resistant to "link spam" - a process of creating artificial web pages that link to some main page with the goal of boosting that page's ranking.

However PageRank has some drawbacks, it is weak when encountering dead end links, where there are no more outgoing links from a certain web page. Another similar problem is rank sinks - when a set of pages gets caught in a link cycle as shown below.



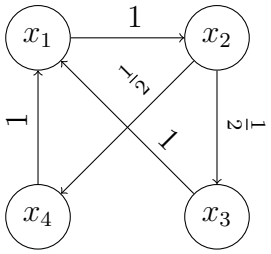
But these problems are much less severe and there are different PageRank tricks that go out and about to overcome them. And we will explain them in the PageRank theotry.

4 Theory behind PageRank

4.1 The Basics

As we have already mentioned multiple times, PageRank ranks a web page based on how many links from other important web pages point to it. But how exactly is this ranking computed? Suppose we have the following graph G with the nodes being the web pages and the edges being the links.

$$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \qquad \text{(1)}$$



Each node starts with 1 "point" and if a node has more than 1 outgoing links, that point is equally distributed among the links to other nodes. In the real PageRank this 1 point is actually converted into $\frac{1}{n}$ where n is the number of total nodes in the graph in order to get a Probability Distribution of each node's importance. In a way, this can be perceived as a Markov Chain. So we can derive a formula for a ranking of a singular web page as the following:

$$PR(v_i) = \sum_{v_j \in Q_v} \frac{PR(v_j)}{L(v_j)}; \qquad \text{(2)}$$

Where the value for a web page v_i is dependent on the ranks of each page v_j in a set of all web pages Q_u divided by the number of outgoing links from each page v_j. But this example graph is a strongly connected graph with no sinking nodes and only one component. But the Web is not strongly interconnected.

4.2 The Surfer Model

PageRank handles this using a model of an unbiased surfer who clicks on links through web pages randomly, but there is a probability that this surfer will reset from any random web page in the Web at every step. This probability is called the damping factor(d) and is usually taken to be 0.85, meaning that there is an 85% chance for a random surfer to click a link from a web page that he is currently on and 15% chance to go to any random web page on the whole web, meaning he can enter a different component of the graph, which is not connected to the previous web page graph he was on, effectively solving the problem of being locked in a set of web pages. Hence, our formula now looks like this:

$$PR(v_i) = \frac{1-d}{n} + d \sum_{v_j \in Q_v} \frac{PR(v_j)}{L(v_j)}; \tag{3}$$

As you can see, when the surfer is reset, is takes a random page out of all the n web pages without any bias.

4.3 The Google Matrix

However, there is still more you can do. The web page graphs that we can work with are not as ideal as the one that we give in the example 1. That is when the idea of a google matrix comes in. Google matrix is a stochastic matrix which connects the sink nodes to all the other nodes in a graph in other words, the columns having zeros are replaced with $\frac{1}{n}$, effectively removing the previously specified sinking problem during ranking. It can be built from a simple adjacency matrix containing ones and zeros and it normalizes all the values so that the sum of each columns is equal to unity.

That is the actual terminology!

4.4 The dominant Eigenvector

With all of that knowledge at our arsenal we can now make PageRank easier for iterative computation on our machines. Since we now have a Google matrix, we can start using Markov Chain Theorems on it, because the matrix is stochastic. Moreover, the matrix is irreducible and aperiodic, which means that it admits a stationary distribution, that is:

$$\pi = P\pi; \tag{4}$$

After an infinitely long "travel" of the surfer, the probability distribution will converge to a stationary distribution π. In this equation π is also the eigenvector of P corresponding to the eigenvalue $\lambda_1 = 1$. And according to the Frobenius-Perron theorem, if matrix P is positive, then it has a positive eigenvalue p such as $|\lambda_x| < p$, where λ_x is an eigenvalue of P, then the eigenvector π of P with eigenvalue p is considered a unique positive eigenvector. Our case satisfies the Frobenius-Perron theorem and we conclude that π is the dominant eigenvector of P with the dominant eigenvalue 1. This allows for transforming our formula into the following:

$$\mathbf{R} = \begin{pmatrix} PR(v_1) \\ PR(v_2) \\ \vdots \\ PR(v_n) \end{pmatrix} = \begin{pmatrix} \frac{1-d}{n} \\ \frac{1-d}{n} \\ \dots \\ \frac{1-d}{n} \end{pmatrix} + d \begin{pmatrix} r(v_1,v_1) & r(v_1,v_2) & \dots & r(v_1,v_n) \\ r(v_2,v_1) & \ddots & & \vdots \\ \vdots & & r(v_i,v_j) & \vdots \\ r(v_n,v_1) & \dots & & r(v_n,v_n) \end{pmatrix} \mathbf{R} \tag{5}$$

Where $\mathbf{R}$ is the dominant eigenvector and the function $r(v_i, v_j)$ is the ratio of the links going from web page j to web page i to the total number of links going out from web page j. This function returns 0 if there are no connections between web pages i and j and is normalized, i.e:

$$\sum_{i=1}^{n} r(v_i, v_j) = 1 \tag{6}$$

4.5 Computation Methods

From the previous section, we know that we need to compute the dominant eigenvector of a matrix to get all the PageRanks. This is called the Eigendecomposition method and it is slow, having the complexity of $O(n^3)$. Instead, another, more effective method is used, called the Power Method.

The Power Method utilizes the fact that our PageRank vector is a dominant eigenvector, meaning we can easily compute it iteratively, by initializing this vector as random(just having values $\frac{1}{n}$) and updating it at each iteration by computing $v = P * v$.

Power Method has complexity $O(n^2)$, but since our matrices are sparse, that is, containing a lot of 0's, it can be reduced to $O(n)$ with the sparse matrix multiplication, $O(i * n)$ exactly, where i is the number of iterations of the method.

The full algorithm is in the next section!

5 Implementation Pipeline

5.1 Tools Data

All the tests and algorithm implementations are written with Python programming language with the help of numpy library. For the performance comparison with already-made solutions, we took the PageRank implementation from the networkx library, which was also used for generating artificial data for testing.

The real test data was originally supposed to be taken from a custom web crawler written in Rust for the Architecture of Computer Systems project by Bohdan Ruban and Mykhailo Bondarenko. However due to some minor complications with that project, we decided to take an already scraped dataset of websites from Hollins University [1, on Github].

5.2 Structure

All of our implementations are in the pagerank.py module. It contains the method for getting the google matrix, the simple PageRank which computes the eigenvector directly, and 2 power methods, one with ϵ (which is the error between the last 2 iterations) that helps us understand when the iterations don't give any further noticeable improvement, but uses array copy at each iteration which slows it down. And one with simply a number of iterations hard-coded, which will be more effective when the number of iterations is taken to be ideal, because it does not utilize array copying.

Since we are all about speed and effective programming here in Ukrainian Catholic University, we could use another python library - numba, for speeding up python. Or we could have just chosen C++ to write this project...

5.3 Google Matrix Conversion Pseudocode

To start writing PageRank, we need to have a Google Matrix converter at first. Here is how it looks:

Data: G: adjacency matrix; d: the damping factor
Result: The Google Matrix

$A \leftarrow G.Transpose$
$n \leftarrow A.size$
$last \leftarrow A[0]$
while *last exists* **do**
  if *last is a column of 0's* **then**
    replace each element with 1/n;
  else
    skip;
  end
end
Build transition matrix D for G into google matrix;
$G \leftarrow dot(A * D)$
$G \leftarrow d * G + (1 - d) * vec(\frac{1}{n})$
return G;

Algorithm 1: Get Google Matrix

As you can see, we apply the damping factor in the end of the conversion. Most algorithms apply the damping factor during PageRank, but we decided it would be more comfortable for cleaner PageRank implementations to put it here.

5.4 Simple Eigendecomposition Pseudocode

This method just computes the dominant eigenvector of a matrix - $O(n^3)$:

Data: G: Adjacency Matrix
Result: The PageRanks in an eigenvector

Convert the matrix into Google Matrix;
Compute the eigenvectors directly(via libraries);
Get the eigenvector corresponding to the largest eigenvalue;
Normalize it;
Return the normalized dominant eigenvector;

Algorithm 2: Eigendecomposition Method

Note: This method is simply here so we can compare it to the Power Method and show how much slower computing the eigenvectors directly actually is.

5.5 The Power Method Pseudocode

The Power Method is the most efficient method of computing PageRank. It utilizes the fact that the PageRank vector is the dominant eigenvector.

Has variations that are more efficient!

Data: G: Adjacency Matrix; num-iter: Number of iterations; e: epsilon, the "halt" difference
Result: The PageRanks in an eigenvector

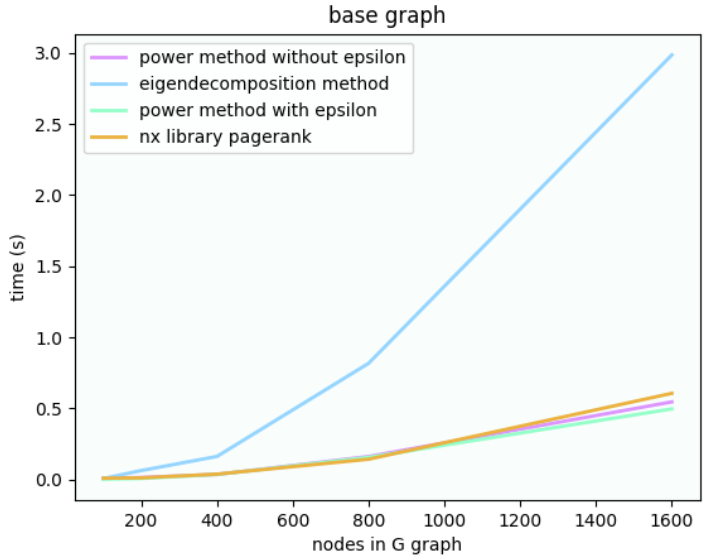
$G \leftarrow GetGoogleMatrix(G)$
$V \leftarrow vec(\frac{1}{n})$
for i in range num-iter do
$pV \leftarrow V.copy()$
$V \leftarrow dot(G * V)$
if $abs(pV - V)$ ¡ e **then**
  break;
else
  skip;
end
end;

Algorithm 3: The Power Method

6 Results

6.1 Artificial Data

First we test our implementations on the artificial data, generated with the help of the networkx library. We tested 4 implementations, 3 of which are ours: eigendecomposition, power method with "halt" epsilon error, power method which simply runs for a fixed number of iterations and nx.pagerank - a PageRank implementation from the networkx library. Here are the results:



The test graph was generated like this: each node in a graph had $\frac{1}{n}$ edges, where n is the total number of nodes in the graph.

As you can see, the eigendecomposition method is the slowest one here, with it's complexity $O(n^3)$. Then come the power methods, as well as the networkx library method, having time complexity $O(V * N)$, where V is the amount of iterations of the power method. And our implementations are even somewhat faster than the networkx implementation! :D

Note: In this case, all the PageRanks produce the same results with the accuracy of epsilon $\epsilon = 10^{-8}$. However some PageRank modifications include sacrificing the accuracy for speed.

6.2 Real Data

As it was already mentioned, all of the pre-crawled data was taken from the Hollins University, where they had all of their web pages already scraped. The dataset has 6012 web pages and can be found in the data section in the github[1] of your implementation(in.txt file). The sorted output ranking can be found in the out.txt file. Here are the top 3 entries after running our implementation of the power method.

```
http://www.hollins.edu/ 0.01182240
http://www1.hollins.edu/faculty/saloweyca/clas%20395/BronzeAGe/sld008.htm 0.01007654
http://www1.hollins.edu/docs/academics/international_programs/forms.htm 0.00803295
```

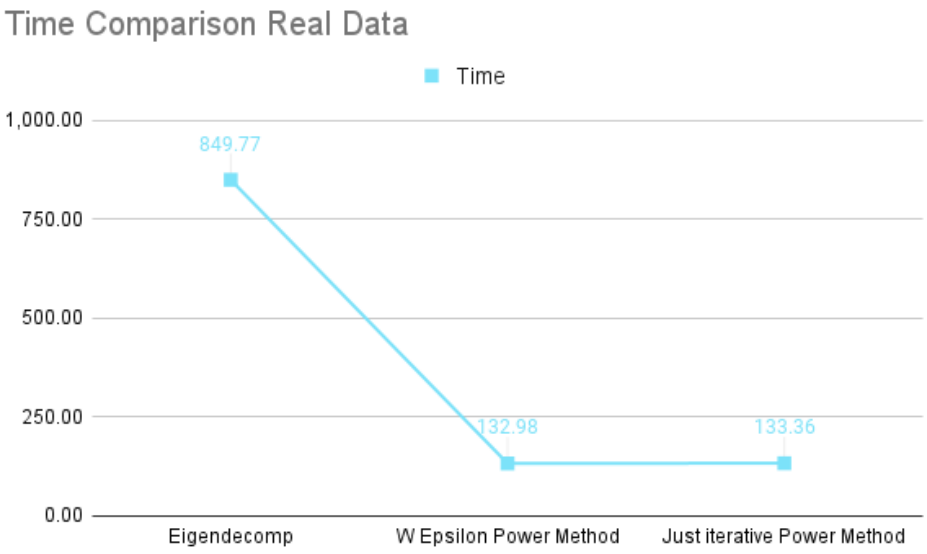
As is clearly seen, the main university page got the best result: $PR = 0.011$, which is logical because it is being linked to the most. The second and third place was given to the less important pages like international programs.

Note: unfortunately, since our main source of data - the previously mentioned Rust Web Crawler experienced some problems right before defence, this data set was taken in a hurry, that is why it is quite old, but it is still sufficient enough to show that our PageRank implementation works correctly!

6.3 Real Data Speed Test

At last, we decided to test and compare the speed of all 3 of our implementations and as we can see, both Power Methods are working at around the same time of around 130 second, while the eigendecomposition method took a whole 850 seconds to compute.

Note: Here the ranking slightly differ from method to method as they possess different accuracy and trade it for speed(Power Method).



7 Conclusion

In this work, we discusses Google's PageRank and how it operates on the web, giving us the best possible results for our search queries. We discussed PageRank and HITS ranking algorithms and why one has an edge over the other.

We also learned that PageRank, as well as other ranking algorithms, consider the Web to be a big graph with a bunch of nodes as web pages and edges as links. And how PageRank computes the importance of a web page without knowing anything about it and only the number of links between pages is fascinating.

We also implemented a couple of ways for computing PageRank like Eigendecomposition or The Power Method and compared their efficiency to the built in solutions. We also tested our implementations on the real data from a (unfortunately not ours) web crawler.

All in all, we had fun with it, even despite our crawler failure. Our implementations turned out to be as effective as the already available ones and give us a correct result.

References

[1] https://github.com/iamthewalrus67/pagerank

[2] https://towardsdatascience.com/pagerank-algorithm-fully-explained-dc794184b4af

[3] https://towardsdatascience.com/hits-algorithm-link-analysis-explanation-and-python-

[4] Altman, Alon; Moshe Tennenholtz, *Ranking Systems: The PageRank Axioms*, 2005.

[5] Page, Lawrence, *Original PageRank U.S. Patent—Method for node ranking in a linked database—* , 1998.

[6] Nidhi Grover, *Comparative Analysis Of Pagerank And HITS Algorithms*, 2012.