21-241 Final Project

(Oswaldo)oswaldor
(Matthew)mbshao

December 1, 2022

1 Introduction

Nowadays, when looking for information, most people turn to the internet as instincts would direct them. The best way for someone to find an answer to their questions is to go online and look it up through search engines like Google, Firefox, Safari, etc. But how do these search engines know how to prioritize the most optimal solutions? In order to successfully do a search, it is necessary for a search engine to have a system to prioritize the significance of results (in this case webpages). Without this, we would be bombarded with websites and information that could be completely irrelevant to the information we want to find. Although the logic behind this system is quite straightforward, there are still a few defining criteria that are harder to explain. When attempting to address this issue, many questions are not necessarily simple to explain, such as how do we actually evaluate the significance of a page? How can we contrast the significance?

In this essay, we will describe and examine each of these elements in terms of how they contribute to the PageRank algorithm and the essential linear algebra for the entire procedure.

2 PageRank Preview

Let's first start by giving some insight of what the PageRank algorithm is, how it came to be and how it functions. In 1998, this algorithm was developed by Sergey Brin and Larry Page who stated that the "rank" of a page could be classified by how many other pages or sites pointed at it. This is known as "link popularity." Link popularity calculates the rank of a "page" by weighing the amount of the other pages that point to it and the amount of pages that the page points too. In this sense, we can imagine link popularity as a graph of nodes and edges where each node would represent a page, and the "edges" would represent the connections between the nodes. However, link popularity does not come without faults. If it is the case that we reach a node/page that does not point to any other page/node, we have reached a sink state(a node with an out degree of 0). To combat this, PagerRank incorporates a damping factor which prevents the failure of the PageRank algorithm in the case that a sink state is reached by allowing a degree of randomness in which for a small percentage of the time, we do not follow the edges of the node we are at and instead jump to another node in the graph.

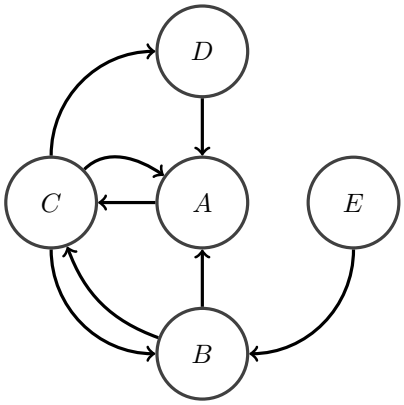
3 Mathematics of the PageRank Algorithm (Markov)

3.1 What is Markov?

Markov chains and matrices are the stochastic model which describes the sequence of probable events, where the probability of the next event depends strictly on the current event. Meaning, information about past events or operations is negligible in terms of determining the probability of what will happen next.

3.2 How does this relate to the PageRank Algorithm?

In the bigger picture, let's assume there are n webpages where each webpage has a list of k (in between 0 and $n-1$) webpages that the webpage directly links to. We want to perform the PageRank Algorithm on the n pages using each webpages' list of webpages. Since Markov chains measure the probability of a sequence of events, the PageRank Algorithm measures the probability of going from one webpage to another. In this case, the Markov chain can be visualized as a graph with n nodes that also contain the probabilty of going from one of the nodes to another. If we were to "randomly walk" from one page to another a million times, there will definitively be some pages that get visited more than others. The pages that get visited more often will be the ones that are ranked higher through this algorithm.



3.3 Visualization

To better understand this concept, imagine there are 5 webpages graphed out with a Markov chain. As an example, page E points to page B which means there is a link in page E that directs you to page B and so on for all the arrows. Since pages like page C have more than one arrow going out, i.e. to A, B, D , this means that there is an equal probaility of 1/3 a random walk out of C.

This depicts a very general case of a directed graph in which walking from one page to another is always possible. We will account for the case where we end up in a sink state later.

$$M = \begin{pmatrix} 0.0 & 0.5 & 0.333333 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.333333 & 0.0 & 1.0 \\ 1.0 & 0.5 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.333333 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{pmatrix}$$

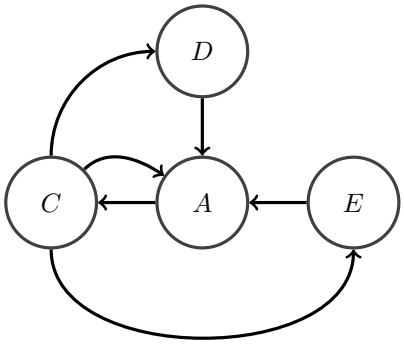
Rather than performing an iterative method on the Markov matrix, we instead find the eigenvectors of the matrix. As a key property of positive Markov matrices, the largest eigenvalue is $\lambda = 1$, and the corresponding eigenvector is called the "attracting steady state." As you perform more and more random walks, the probabilities of the person (or in our case, the user) will be in a state converges to a value, and the vector of these values is the steady state. Our PageRank algorithm takes advantage of this fact to compute the rankings of each page. In our toy example, running the PageRank algorithm results in

$$\begin{pmatrix} 0.33 \\ 0.13 \\ 0.4 \\ 0.13 \\ 0 \end{pmatrix}$$

3.4 Sink State Adjustment and Damping Factor

Coming back to the issue of a sink state, we want to strengthen our algorithm such that if a user reaches a sink state, they're able to still continue randomly walking the network. To incorporate this feature, we made it so that if a user reaches a sink state, they will be transferred to any of the other webpages with equal probability.

Now, consider the question: What if the user stops clicking? This is where the damping factor plays a role. The damping factor, conventionally set to 0.85, is the probability that the user will stop clicking. In the iterative version of PageRank, the computation for each webpage's rank would include the damping factor because it could be the case that the user stops clicking. Unfortunately, it's difficult to add a damping factor to our algorithm because we're instantly computing the steady state, which is derived from running large numbers of random walks. In addition, our algorithm is not iterative. This is important to consider because no real person infinitely roams the internet!



4 HITS Algorithm

4.1 What is HITS?

An overview of HITS is that it is a link analysis algorithm that ranks webpages based on two factors; authority and hub. A page's authoritative rank estimates the page's content vs a page's hub rank measures its links(edges) or connections to other pages.

4.2 HITS Visual Representation

Let there be 4 pages, A,C,D, E and let the arrows indicate a connection from one page to another. Note that these connections are one way so with the case of page C pointing to page D, page D does not necessarily point or redirect back to point C. From the graph it is clear there is no arrow from page D back to C.

	O	O	N	N
A	1	1	0	0
C	1	1	0	0
D	1	1	0	0
E	1	1	0	0

For simplicity of reading, let the first O, N columns represent the old authority and new authority scores respectively. The same applies for the 2nd O and N columns, however, they represent the old and new hub scores respectively.

	O	O	N	N
A	1	1	3	1
C	1	1	1	3
D	1	1	1	1
E	1	1	1	1

To find the new authority scores for our pages, we have to look at the in-degree of each page. The in-degree of a page is the number of other pages that point to that page. For our example diagram, pages C, D, E all point to A. Since all 3 of them have a hub score of one, we get 1+1+1 which makes the new authority score of A = 3. Next, we do the same for pages C, D, and E.

Now to calculate the new hub score for the pages we look at the out-degree of the pages. The out-degree is the number of other pages a single page is point to. For example, page A is pointing to 1 page, page C. Now to calculate A's new hub score we take each page it's pointing to and add the old authority score of the page it points to. In this case its 1 so A's new hub score is 1. The same is done to calculate the hub score for nodes C, D, E.

	O	O	N	N
A	.5	.16		
C	.16	.5		
D	.16	.16		
E	.16	16		

Next, the grid needs to be normalized before the next iteration. For this to be done, the total number each page's new authority and new hub scores must be added up and divided into the new authority and new hub scores. For this iteration, there is a total of 6 for the new authority score so A's score before .5 after division. Same occurs for the hub scores; after all arithmetic is done the new authority and hub scores become the old scores and the calculations continue. The new chart should look like this. (Since the N cols will be overwritten anyway after the next iteration, assume they are blank.)

4.3 Extracting and analysis of the "PageRank" from HITS

Without having to compute each step here, Julia helps by simplifying all the calculations. After all this mindless computation finished; to extract the rank, we averaged the authority and hub scores of each page and ranked their average.

```
T = [[2],[1,3,4],[1],[1]]
HITS(T)
✓ 0.1s
4×4 Matrix{Float64}:
 0.498559    0.00288193  0.499038    0.00192313
 0.00288193  0.498559    0.00192313  0.499038
 0.24928     0.24928     0.249519    0.249519
 0.24928     0.24928     0.249519    0.249519
```

The image has the same layout as the grids mentions above where the first two cols record the old authority and hub respectively and the 3rd/4th cols record the new authority and hub scores respectively. We set our HITS algorithm to end when the difference after an iteration is smaller than $+-0.001$.
We observe the 3rd/4th columns which provide the Authority and Hub scores for our pages A, C, D, and E.

Rankings:
Page(A) Rank 1 (Average .25048)
Page(C) Rank 1 (Average .25048)
Page(D) Rank 3 (Average .24952)
Page(E) Rank 3 (Average .24952)

From the results we can see that pages A and C both tie for rank 1 even though A has a high authority score and C has a high hub score.

Pages D and E tie for 3rd and 4th but are not too far off of pages A and C. This results from the fact they have pretty average authority and hub scores. From the way we used HITS to rank them it does not solely depend on how good a page's authority is, for the rankings.

4.4 Analysis and Comparison between PageRank and HITS

The differences between the PageRank algorithm and the HITS algorithm is noticeable. For one, both algorithms have a different criteria for the ranking. HITS looks at the Authority score and Hub score of each webpage, while PageRank only considers the out-degree of each webpage. Because of this difference, PageRank discriminates against hubs while favoring authorities. However, for our project, we took the average of the two scores to compare the results of the PageRank algorithm. As aforementioned, HITS gave a more holistic ranking of the webpages, so HITS was more fair to each webpage. Both algorithms have their respective advantages depending on the circumstance, but Google uses PageRank so they should take a look into the HITS algorithm.

5 Conclusion

In summary while working on this project we learned that coding in Julia is relatively similar to coding in Python. Neither of us had extensive knowledge of coding in either language prior to this assignment but we learned that with our the good old friend the internet anything is possible. We also found it interesting how PageRank algorithms enabled us to find the resources we needed to efficiently complete this project.

We also learned more about the usefulness of Markov Chains and Markov matrices. By creating visual representations of the probability distributions, we were able to gain further insight and understanding of how PageRank and HITS work.

6 PageRank on Large Network

After testing our implementation of the PageRank algorithm on a simple web graph, we were interested how our algorithm would fair against a larger web graph. Supplied by the Stanford Large Network Dataset Collection, we had access to Google's web graph with 875713 nodes and 5105039 edges. The web graph was mighty, but Julia was mightier! However, we weren't able to actually test our implementation on the web graph because we had trouble converting the data into a valid input for pageRank. Moreover, most of the times we ran into memory issues.

7 Julia Code

7.1 PageRank Algorithm

```
function findMarkov(L)
    #Create Markov matrix from the 2D list of websites
    size = length(L) #Find the size of the list
    M = zeros(size,size) #Create empty matrix
    col = 1 #Start at the first webiste
    for web in L #Iterate through the websites
        if length(web) == 0 #Is sink state
            for i in 1:size
                M[i,col] = 1/size #randomly place it anywhere in the graph
            end
            col += 1
            continue
        end
        for subweb in web
            M[Int(subweb),Int(col)] = 1/length(web) #Set the entry to its transition probability
        end
        col += 1 #Increment col
    end
    return M
end
function pageRank(L) #where L is a 2D list of websites and their links
    M = findMarkov(L) #Find the Markov Chain associated with the 2D list
    X = eigvecs(M) #Find the eigenvectors of the Markov Chain
    x = X[:,size(X)[1]] #Find the eigenvector with eigenvalue 1
    res = (1/sum(x))*x #Scale it so that the sum of the values is 1
    return res
end
```

7.2 HITS Algorithm

```
function HITS(L)
    closeenough = false
    diff = .001
    firstitr = true
    size = length(L)
    M = ones(size,4)
    while (true)
        inc = 0
        for web in L
            inc+=1
            for subweb in web
                M[subweb,3] += M[inc,2]
                M[inc,4] += M[subweb,1]
            end
        end
        if (firstitr == true)
            for i in 1:length(L)
                if (M[i,3] == 1)
                    M[i,3] = 0
                end
                if (M[i,4] == 1)
                    M[i,4] = 0
                end
            end
        end
        sumNewAuth = 0
        sumNewHub = 0
        for i in 1:length(L)
            sumNewAuth += M[i,3]
            sumNewHub += M[i,4]
        end
        for i in 1:length(L)
            M[i,3] = M[i,3]/sumNewAuth
            M[i,4] = M[i,4]/sumNewHub
        end
        counter = 0
        for i in 1:length(L)
            if (M[i,1] - M[i,3] > diff)
                counter+=1
            end
            if (M[i,2] - M[i,4] > diff)
                counter+=1
            end
        end
        #If the difference is close enough then start ranking them
        if (counter < 1)
            res = zeros(1,size)
            for i in 1:size
                res[i] = (M[i,3] + M[i,4])/2
            end
            return M
        end
        #Now set the old to new if the lambda is not reached
        for i in 1:length(L)
            M[i,1] = M[i,3]
            M[i,2] = M[i,4]
        end
    end
end
```

7.3 Testing PageRank on Larger Network

```
#Helper functions for converting the information from textfile into a webgraph and valid input
using DelimitedFiles
function string2Int(s)
    size = length(s)
    k = 1
    notFound = true
    while notFound
        if s[k] == '\t'
            notFound = false
            break
        end
        k+=1
    end
    (parse(Int,s[1:k-1]),parse(Int,s[k+1:size]))
end
function convert2List(A)
    final = []
    temp = []
    len = size(A)[1]
    for i in 1:len
        if i == len
            append!(final,(A[i,1],temp))
            continue
        end
        if A[i,1] != A[i + 1,1]
            append!(temp,(A[i,2]))
            append!(final,(A[i,1],temp))
            temp = []
            continue
        end
        append!(temp,(A[i,2]))
    end
    return final
end

#Converting the Google webgraph into valid input for pageRank
webGraph = open("web-Google.txt")
lines = readlines(webGraph)
res = zeros(size(lines)[1] - 4, 2)
for i in 5:(size(lines)[1])
    (res[i - 4,1],res[i - 4,2]) = string2Int(lines[i])
end
close(webGraph)
L = convert2List(res)
nodes = []
edges = []
for i in 1:length(L)
    if i % 2 == 1
        append!(nodes,L[i])
        continue
    end
    append!(edges,[L[i]])
end
#Run pageRank on the webgraph's edges
pageRank(edges)
```

```
# Directed graph (each unordered pair of nodes is saved once): web-Google.txt
# Webgraph from the Google programming contest, 2002
# Nodes: 875713 Edges: 5105039
# FromNodeId  ToNodeId
0 11342
0 824020
0 867923
0 891835
11342 0
11342 27469
11342 38716
11342 309564
11342 322178
11342 387543
11342 427436
11342 538214
11342 638706
11342 645018
11342 835220
11342 856657
11342 867923
11342 891835
824020  0
824020  91807
824020  322178
824020  387543
824020  417728
824020  438493
824020  500627
824020  535748
824020  695578
824020  867923
824020  891835
867923  0
867923  11342
867923  136593
867923  414038
867923  500627
867923  523684
867923  760842
867923  815602
867923  835220
867923  846213
867923  857527
867923  891835
891835  0
891835  11342
891835  112028
891835  235849
891835  302284
891835  417728
891835  451592
891835  693969
891835  857527
891835  867923
1 53051
1 203402
1 223236
1 276233
1 552600
1 569212
1 635575
1 748615
...     ...
916425  116832
916425  197330
```

8 References

Wikipedia contributors. (2022, November 22). PageRank. In Wikipedia, The Free Encyclopedia. Retrieved 23:18, December 9, 2022, from https://en.wikipedia.org/w/index.php?title=PageRankoldid=1123190328

Strang, G. (2016). Introduction to Linear Algebra (Gilbert Strang) (5th ed.). Wellesley-Cambridge Press.

Mohit. (2018, October 16). HITS Algorithm and HUBS and AUTHORITIES Explained [Video]. YouTube. https://www.youtube.com/watch?v=-kiKUYM9Qq8

SNAP: Stanford Network Analysis Project. (n.d.). https://snap.stanford.edu/