21-241 Final Project

December 02, 2021

Brady Wales, SCS 2025

Shaheer Aslam, SCS 2025

PageRank using Markov Matrices

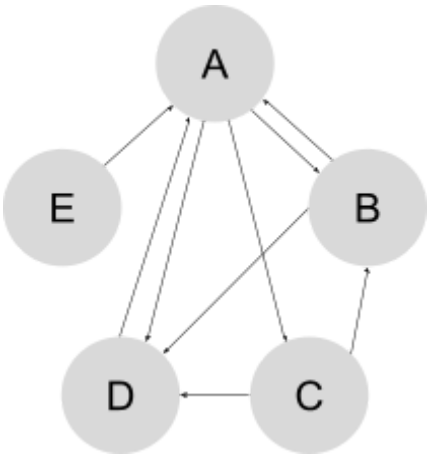
How Google Works

Overview

PageRank is an algorithm developed by Google co-founder Larry Page used to rank websites. The algorithm works by ranking pages based on the number of links pointing towards that page while also taking into account the weight of the websites pointing to the page. A website's weight is based on the weight of the websites pointing to it.

Goal

Knowing the links to and from each page in a set of pages, calculate the rankings of the pages so that they can be ordered as search results.



An example visualization of pages with arrows representing links pointing to other pages.

Steps of PageRank algorithm

- For each page, get data on which pages it links to.

  This is done by storing a vector of bits where a 1 represents there being a link to the page in that index of the vector, page indexes within the vector are decided arbitrarily and the order of indexes does not affect the outcome. That being said for our representation we will be using a vector with indexes ⟨a,b,c,d⟩. Index a will represent there being a link to page A, index b represents a link to page B, and so on.

  i.e. In the example above:

  Vector A: ⟨0, 1, 1, 1, 0⟩ Vector B: ⟨1, 0, 0, 1, 0⟩

  Vector C: ⟨0, 1, 0, 1, 0⟩ Vector D: ⟨1, 0, 0, 0, 0⟩

  Vector E: ⟨1, 0, 0, 0, 0⟩

- Normalize vector such that the sum of all nonzero entries equals 1.

  This is done to represent the probability of a random website being chosen from the possible links.

  i.e. The vector representing Page A from above would now be $\langle 0 , \frac{1}{3} , \frac{1}{3} , \frac{1}{3} , 0 \rangle$

- Create a Markov Matrix L with columns being made up of the vectors representing the pages. This will be our Link Matrix.

  A Markov Matrix allows us to view and use the information to find long-term patterns over multiple iterations of data manipulation. The data is read by each column illustrating the different pages and each row representing the page it points to. The example from above written as a Markov Matrix is below.

	A	B	C	D	E
A	0	$\frac{1}{2}$	0	1	1
B	$\frac{1}{3}$	0	$\frac{1}{2}$	0	0
C	$\frac{1}{3}$	0	0	0	0
D	$\frac{1}{3}$	$\frac{1}{2}$	$\frac{1}{2}$	0	0
E	0	0	0	0	0

- Create a vector r, with a length of the number of pages you are working with. In each entry in the vector, set the value to $\frac{1}{Number\ of\ pages}$.

  This vector will represent the rankings and weight of each page. It is initialized with the same value for each page.

  For our example, r would be initialized to: $\langle \frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5} \rangle$

- Multiply L*r a number of times until eventually r converges and the values no longer change.

  By doing this you are adjusting the weight of each page depending on the pages which have links pointing to it. Eventually, this vector will converge and the resulting vector will be the weight of each page in decimal form. This vector is in steady-state form. By properties of Markov Matrices, this steady-state vector is also the eigenvector when $\lambda = 1$.

  Our example vector would converge to $\langle 0.387, 0.193, 0.129, 0.290, 0.0 \rangle$

- Finally, by having the decimal weights of each page, you can create a ranked list in order based on the weight of each page.

By sorting, the PageRank would be [A, D, B, C, E].

Extension - HITS algorithm

The Hyperlink-Induced Topic Search (HITS) algorithm developed by Jon Kleinberg is another way of ranking pages based on links to and from other pages. It differs from PageRank by the idea that sites can have both an authority score and a hub score instead of just weight. A site will have a high hub score if it has links to many other sites, while a site will have a high authority score if is linked by many hubs.

Steps of HITS algorithm

- To start, like with PageRank, get data on links from each site.

  We use the same process of storing information in vectors. Using the same example we have:

  Vector A: <0, 1, 1, 1, 0> Vector B: <1, 0, 0, 1, 0>

  Vector C: <0, 1, 0, 1, 0> Vector D: <1, 0, 0, 0, 0>

  Vector E: <1, 0, 0, 0, 0>

  Also, create a matrix M with each vector as a column of the matrix. Like a Markov Matrix, the columns represent each page and the rows represent where the page links to.

	A	B	C	D	E
A	0	1	0	1	1
B	1	0	1	0	0
C	1	0	0	0	0
D	1	1	1	0	0
E	0	0	0	0	0

- Create initial vectors with the length being the number of pages to represent the authority and hub score for each page. We start with a hub and authority score of 1 for each page.

  For our example with five pages, these vectors, a and h, would both be = ⟨1, 1, 1, 1, 1⟩

- For each page, update its hub and authority score.

  First, update the hub score. To do this, we calculate the sum of the authority score of all pages that the page points to. Using the dot product of the page vector and the authority vector gives us our hub sum.
  i.e.

  $$A \bullet a = 3$$

  Therefore, our initial hub score of page A is 3 as it links to 3 pages. Now set the entry h_1 (where 1 will be the entry in the vector corresponding to A, 2 corresponds to B, and so on) to this score, 3.

  Then update the page's authority score. To calculate the authority score, simply calculate the sum of the hub scores of all pages which point to it. This is done using the dot product of the row of matrix M corresponding to the page you are updating with your hub scores vector.
  i.e.

  $$M_A \bullet h = 3$$

  Our initial authority score of page A is also 3 as it is linked to by 3 pages. Set the entry a_1 (using the same system of storing pages in order as the hub vector) to be this score, 3.

  After doing this for each page, our vector h will equal ⟨3, 2, 2, 1, 1⟩ and vector a will equal ⟨3, 2, 1, 3, 0⟩

- After calculating the hub and authority score for each page, it is required to normalize the vectors otherwise the values will diverge.

  Normalizing h will result in $\langle \frac{3}{\sqrt{19}}, \frac{2}{\sqrt{19}}, \frac{2}{\sqrt{19}}, \frac{1}{\sqrt{19}}, \frac{1}{\sqrt{19}} \rangle$

  Normalizing a will result in $\langle \frac{3}{\sqrt{23}}, \frac{2}{\sqrt{23}}, \frac{1}{\sqrt{23}}, \frac{3}{\sqrt{23}}, 0 \rangle$

- Repeat the past two steps a finite number of times until eventually, the values in h and a will converge.
  Eventually,

  a will converge to ⟨0.124, 0.319, 0.208, 0.916, 0⟩

  h will converge to ⟨0.355, 0.437, 0.77, 0.211, 0.211⟩

This means page A has an authority score of 0.124 and a hub score of 0.355. Going through the entries gives the authority and hub score of each page.

- Now that we have authority and hub score for each site, we can display the sites in order based on the search query.

  For example, imagine a user is looking to buy a used car. If they don't know what car they want and search "used cars," the sites should be displayed based on the hub score as a good hub would have many links to pages of different listings of cars for sale. Now, if the user knows what type of car they want and search for it, pages with high authority should be displayed first as these pages will be the listings with links pointing to it from many hubs.

Conclusion

By knowing a set of pages and their links, both PageRank and HITS algorithms can be used to rank pages. These rankings are used by search engines in order to give the user a list of pages relevant to their search topic and ranked based on the quality of the sites.

Julia Notebook and References

Markov Chains, Random Walks and PageRank

The purpose of this notebook is to explore the algorithm of PageRank, developed by Google, and create toy examples while exploring extensive concepts like Markov matrices, probability and ranking websites.

Importing Libraries

Here we import all necessary libraries.

In [1]:
```
using LinearAlgebra
```

Processing

We create algorithms to process and conduct a mini PageRank.

In [3]:
```
# function userInput which gets user input, processing into L matrix
function userInput()
    # getting number of sites
    #println("How many websites do you have?")
    #websites = readline()
    #websites = parse(Int, websites)
    println("You have five websites.")
    websites = 5
    println("Format: 2 3 4 (website 1 points to 2, 3, 4)")

    v1 = []
    v2 = []
    v3 = []
    v4 = []
    v5 = []

    # iterating through sites
    for i in 1:websites
        # getting links each site points to
        print("For website number ")
        print(i)
        println(" input the links which it points to.")
        links = readline()
        # converting input to array
        arr = split(links, " ")
        tempVector = zeros(websites, 1)
        # looping through links
        for j in arr
            number = parse(Int, j)
            # normalizing and inputting into vector
            tempVector[number] = 1/length(arr)
            if i == 1
                v1 = tempVector
            elseif i == 2
                v2 = tempVector
```

```
            elseif i == 3
                v3 = tempVector
            elseif i == 4
                v4 = tempVector
            elseif i == 5
                v5 = tempVector
            end
        end
    end

    L = [reshape(v1, 1, :); reshape(v2, 1, :); reshape(v3, 1, :); reshape(v4, 1,

    return L
end
```

Out[3]:
```
userInput (generic function with 1 method)
```

Using Functions

Here, we assign variables to our functions and use them as such.

In [4]:
```
# this is our L matrix
L = transpose(userInput())
```

```
You have five websites.
Format: 2 3 4 (website 1 points to 2, 3, 4)
For website number 1 input the links which it points to.

For website number 2 input the links which it points to.

For website number 3 input the links which it points to.

For website number 4 input the links which it points to.

For website number 5 input the links which it points to.
```

Out[4]:
```
5×5 transpose(::Matrix{Float64}) with eltype Float64:
 0.0       0.5  0.0  1.0  1.0
 0.333333  0.0  0.5  0.0  0.0
 0.333333  0.0  0.0  0.0  0.0
 0.333333  0.5  0.5  0.0  0.0
 0.0       0.0  0.0  0.0  0.0
```

In [5]:
```
# this is our r vector
r = ones(5, 1)
for i in 1:length(r)
    r[i] = r[i]/5
end
```

Now, we create our Page Rank algorithm.

In [6]:
```
function PageRank(L, r, iterations)
    v = r
    for i in 1:iterations
        v = L * v
    end
    sorted = sort!(copy(v), dims = 1, rev = true)
    ranks = []
```

```
    for i in sorted

        for j in 1:5
            if v[j] == i && j ∉ ranks
                append!(ranks, j)
            end
        end
    end
    return v, ranks
end
```

Out[6]:
```
PageRank (generic function with 1 method)
```

In [72]:
```
# using PageRank with 100 iterations
v, ranks = PageRank(L, r, 100)
# page ranks for each website, top being rank
println(v)
println(ranks)
```

```
[0.3870967741935477; 0.19354838709677386; 0.1290322580645159; 0.290322580645160
8; 0.0]
Any[1, 4, 2, 3, 5]
```

In [47]:
```
# using PageRank function with 200 iterations
v2, ranks2 = PageRank(L, r, 200)
# page ranks for each website, top being rank
println(v2)
println(ranks2)
# can see that the steady state vector v2 does not change with more iterations
```

```
[0.3870967741935477; 0.19354838709677386; 0.1290322580645159; 0.290322580645160
8; 0.0]
Any[1, 4, 2, 3, 5]
```

Extension

We create the HITS algorithm as follows as an extension of what we've learned from PageRank.

In [8]:
```
# function userInput which gets user input, processing into L matrix
function userInput2()
    # getting number of sites
    #println("How many websites do you have?")
    #websites = readline()
    #websites = parse(Int, websites)
    println("You have five websites.")
    websites = 5
    println("Format: 2 3 4 (website 1 points to 2, 3, 4)")

    v1 = []
    v2 = []
    v3 = []
    v4 = []
    v5 = []

    # iterating through sites
    for i in 1:websites
        # getting links each site points to
```

```
        print("For website number ")
        print(i)
        println(" input the links which it points to.")
        links = readline()
        # converting input to array
        arr = split(links, " ")
        tempVector = []
        # looping through links
        for j in arr
            number = parse(Int, j)
            # normalizing and inputting into vector
            append!(tempVector, number)
            if i == 1
                v1 = tempVector
            elseif i == 2
                v2 = tempVector
            elseif i == 3
                v3 = tempVector
            elseif i == 4
                v4 = tempVector
            elseif i == 5
                v5 = tempVector
            end
        end
    end

    return [v1, v2, v3, v4, v5]
end
```

Out[8]:
```
userInput2 (generic function with 1 method)
```

In [9]:
```
# creating another matrix of websites and sites using
all = userInput2()
all
```

```
You have five websites.
Format: 2 3 4 (website 1 points to 2, 3, 4)
For website number 1 input the links which it points to.

For website number 2 input the links which it points to.

For website number 3 input the links which it points to.

For website number 4 input the links which it points to.

For website number 5 input the links which it points to.
```

Out[9]:
```
5-element Vector{Vector{Any}}:
 [2, 3, 4]
 [1, 4]
 [2, 4]
 [1]
 [1]
```

In [19]:
```
function hits(all, iterations)
    authority = [1.0, 1.0, 1.0, 1.0, 1.0]
    hub = [1.0, 1.0, 1.0, 1.0, 1.0]
    # number of iterations
    for i in 1:iterations
        # looping through each vector
```

```
        for j in 1:length(all)
            authoritySum = 0
            hubSum = 0
            # getting authority sum, to be used as hub score
            for k in 1:length(all[j])
                authoritySum += authority[all[j][k]]
            end
            # getting hub sum, to be used as authority score
            for n in 1:length(all)
                if j in all[n]
                    hubSum += hub[n]
                end
            end
            # setting authority and hub scores
            authority[j] = hubSum
            hub[j] = authoritySum
        end
        # normalizing the hubs and authorities
        hubNormalize = 0
        authorityNormalize = 0
        for p in 1:5
            hubNormalize += hub[p]*hub[p]
            authorityNormalize += authority[p]*authority[p]
        end
        hubNormalize = sqrt(hubNormalize)
        authorityNormalize = sqrt(authorityNormalize)
        # setting authority scores and hub scores to normalized scores
        for m in 1:5
            hub[m] = hub[m]/hubNormalize
            authority[m] = authority[m]/authorityNormalize
        end
    end
    return authority, hub
end
```

Out[19]:
```
hits (generic function with 1 method)
```

In [20]:
```
authority, hub = hits(all, 100)
println(authority)
println(hub)
```

```
[0.12416174248946872, 0.31935612933443747, 0.20826633263821837, 0.91608987498385
33, 0.0]
[0.3552126814227838, 0.43716229741559676, 0.7700798682565879, 0.211766467105475
9, 0.2117664671054759]
```

References

- https://en.wikipedia.org/wiki/PageRank#Simplified_algorithm
- http://blog.kleinproject.org/?p=280
- https://www.youtube.com/watch?v=F5fcEtqysGs