SCOTTYRANK.JL: A JULIA IMPLEMENTATION OF PAGERANK AND HITS

SIYUAN CHEN AND MICHAEL ZHOU

ABSTRACT. PageRank is an algorithm famously used by Google to determine the relative importance of different websites for search results. More generally, PageRank and variations of the algorithm can be applied to any directed graph of objects, where one wishes to find the most "important" nodes, as determined by a combination of the number of nodes pointing to it and the number of nodes that it points to. In our implementation of PageRank, Markov Matrices simulating a random walk along the edges of a directed graph were used to determine each node's relative importance. At every step, the PageRank score of a given node would be distributed among the nodes that can be reached through a directed edge outward from the starting node. Our implementation of the HITS variation of the PageRank algorithm added "hub" and "authority" scores, which distinguish between nodes pointing to many other nodes (hubs) and nodes with many other nodes pointing to itself (authorities).

In our project, we implemented both the PageRank and HITS algorithms using Julia to better understand the linear algebra insights behind the two algorithms, and tested them on datasets of varying sizes and densities.

<p style="text-align: center;">Contents</p>

1. Introduction

2. Mathematical Background

Markov Matrices and Random Walks

Markov Matrices and their applications to Random Walks are the foundational linear algebra concepts behind the PageRank algorithm.

Markov Matrices are square matrices with strictly non-negative entries where the sum of entries in every column is equal to 1. Markov Matrices have several key properties that make them especially useful for iterative computing– for a given Markov Matrix M, the following are true:

Note that we are not dealing with positive MM's because we are doing random walks, which assume that at every step, there is a 0 probability that our marker stays at the current node.

3. Algorithms and Computations

Custom Structs

We defined the structs Vertex and Graph to be used in our PageRank algorithms. Vertices were defined as structs with an unsigned integer index, a list of indices of vertices that have directed edges pointing towards V, and a list of indices of vertices that V has directed edges pointing towards, as shown in the code segment below.

For our purposes, we defined a Graph as a struct with the number of vertices and a list of the vertices in the graph sorted by their index.

```
export Vertex, Graph

struct Vertex
  index::UInt32
  in_neighbors::Vector{UInt32}
  out_neighbors::Vector{UInt32}
end

struct Graph
  num_vertices::UInt32
  vertices::Vector{Vertex} # sorted by index
end
```

Reading Graphs and Vertices from Files

The functions read_graph, read_edge_list, and read_adjacency_list are used to read and construct graphs from text files.

The format for an edge list input file is as follows:

```
<num_vertices> <num_edges>
# Directed edge from Vertex with index 1 to Vertex with index 2.
1  2
# More generally,
<Index from> <Index to>
...
```

The format for an adjacency list input file is as follows:

```
<num_vertices>
# Vertices that the first Vertex has directed edges towards
<Index> <Index> < Index > ...
...
```

PageRank Matrix Generation

In our general PageRank algorithm, we will use the PageRank matrix generated from the graph G.

PageRank General Algorithm

   (a) **PageRank - Iterative Variation**

   The first method that we can use to calculate PageRank is the Iterative method.

   We start with the matrix M and the vector $\frac{1}{n}\mathbf{1} = (\frac{1}{n}, \frac{1}{n}, \cdots \frac{1}{n})$, where M is our PageRank matrix generated from the graph, and n is the number of vertices (passed in as num_vertices). The ones vector has n components, and similarly the Markov matrix M is $n \times n$.

   Our result of the iterative method is the vector $M^k(\frac{1}{n}\mathbf{1})$, where k is the value passed in as num_iterations.

```
function pagerank_iteration(num_vertices::UInt32,
   M::Matrix{Float64}, num_iterations::UInt32)

   M_pwr = Base.power_by_squaring(M, num_iterations)
   M_pwr * (ones(Float64, num_vertices) / num_vertices)
end
```

   (b) **Pagerank - Epsilon Variation**

   In the Epsilon method of our PageRank calculations (pagerank_epsilon), similar to the Iterative method, we start with the expression $M(\frac{1}{n}\mathbf{1}) = M(\frac{1}{n}, \frac{1}{n}, \cdots \frac{1}{n})$, where M is our PageRank matrix generated from the graph, and n is the number of vertices (passed in as num_vertices). The ones vector has n components, and similarly the Markov matrix M is $n \times n$.

   Next, instead of multiplying $(\frac{1}{n}\mathbf{1})$ by M a fixed amount of times, we continue multiplying the vector by M on the left until the norm of the difference between the vector $M^{k+1}(1/n, 1/n, ...1/n)$ and the matrix $M^k(\frac{1}{n}, \frac{1}{n}, ...\frac{1}{n})$ is less than the caller-specified limit of Epsilon.

```
function pagerank_epsilon(num_vertices::UInt32,
  M::Matrix{Float64}, epsilon::Float64)

  prev = ones(Float64, num_vertices) / num_vertices
  curr = M * prev
  while norm(prev - curr) > epsilon
    prev, curr = curr, M * curr
  end
  curr
end
```

4. Results

Give a few examples of it running on our test cases and explain why the results are correct.

5. Discussion

6. Conclusion