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.

E-mail addresses: siyuanc2@andrew.cmu.edu, mhzhou@andrew.cmu.edu.
Date: November 2021.

Contents

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.

Frobenius Norms

The Frobenius Norm is used to calculate the norm of matrices in a way similar to vector magnitudes, and is especially useful for defining a notion of closeness between matrices. The Frobenius Norm of a matrix M, denoted by $||M||_F$, is the square root of the summation of the squares of all entries in the matrix.

As an example, for $A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$

$||A||_F = \sqrt{1^2 + 2^2 + 3^2 + 4^2} = \sqrt{30}$

To provide some intuition for further applications of the Frobenius Norm, we will calculate $||A-B||_F$ where $B = \begin{pmatrix} 1 & 1.5 \\ 3 & 4 \end{pmatrix}$.

$$A - B = \begin{pmatrix} 0 & 0.5 \\ 0 & 0 \end{pmatrix}$$

$$||A - B||_F = \sqrt{0^2 + (0.5)^2 + 0^2 + 0^2} = 0.5$$

Thus, we see that the "closer" two matrices are, the smaller the Frobenius Norm of their difference.

In our project, Frobenius Norms are used in the Epsilon variants of the PageRank and HITS algorithms. In these variations, instead of multiplying the Markov Matrices a fixed amount of times, we use Frobenius Norms and keep multiplying the matrices by themselves until the Frobenius norm of

the difference between the matrix M and the matrix MM is less than a specified limit, which is the value of Epsilon.

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

To read and construct graphs from text files, we wrote the functions read_graph, read_edge_list, and read_adjacency_list.

4. Results

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

5. Discussion

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

6. Conclusion

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.