INFORMATION RETRIEVAL (CS F469) ASSIGNMENT 2

Implementation of PageRank Algorithm

-Tarun Raheja

-2015A7PS0106H

In this assignment the PageRank algorithm has been implemented in Python. PageRank is used to rank websites in search results. It works by counting the number and quality of links to a page to determine a rough estimate of how important the website is. The underlying assumption is that more important websites are likely to receive more links from other websites.

ENHANCEMENT :

The code can handle sparse matrices as well due to the way PageRank is computed.

DATASET:

Political blogosphere Feb. 2005

Data compiled by Lada Adamic and Natalie Glance

Node "value" attributes indicate political leaning according to:

0 (left or liberal)

1 (right or conservative)

Data on political leaning comes from blog directories as indicated. Some blogs were labeled manually, based on incoming and outgoing links and posts around the time of the 2004 presidential election. Directory-derived labels are prone to error; manual labels even more so. Links between blogs were automatically extracted from a crawl of the front page of the blog. These data should be cited as Lada A. Adamic and Natalie Glance, "The political blogosphere and the 2004 US Election", in Proceedings of the WWW-2005 Workshop on the Weblogging Ecosystem (2005). It has around 1200 nodes.

ASSUMPTIONS:

- More important websites are likely to receive more links from other websites
- I have taken $\boldsymbol{\beta}$ to be 0.85 and run the iterations 15 times.

MAJOR DATA STRUCTURES USED

- The code is split into two files; a utility file and a PageRank computation file.
- The pageRank.py file implements the PageRank class, which has the following:
  - A graph attribute that stores a NetworkX graph.
  - A variable to store number of vertices in the graph.
  - A variable to store the teleport probability.
  - A variable to store directed/undirected nature of graph.
  - A dict called ranks that stores the current rank values. **This is central to computation of page rank for sparse matrices.**
- The utils.py file has the following data structures:
  - A list for initially storing data from the csv file.
  - A NetworkX graph object that stores the graph that the csv file represents.

PACKAGES USED

- re, sys, math, random, csv are used for taking csv file input.
- NetworkX is used to maintain the graph.
- Scipy and Numpy are used for the matrix computations while graph displaying.
- Igraph and Matplotlib are used for plotting the graph and displaying the result.

CORE FORMULAE USED AND THEORY

- **Construction of matrix M** :

- If page **i** has **n** out-links. Then for each page **j** which has in-link from **i** **M[j][i] = 1/n .**
- **Handling dead ends** : If there are **m** webpages, then for each dead end page **i** **M[j][i] = 1/m** where **j** lies between 0 and **m** .

- **Initializing R vector**: If there are **n** pages R then **R[i] = 1/n** for $0 \le i < n$.
- **Power Iteration method** :
  - For PageRank : Iterate $R(t+1) = \beta\, M.R(t) + \left[\frac{1-\beta}{N}\right]_N$ until $|r(t+1) - r(t)| > \varepsilon$ where N is total no. of pages.
  - For Topic Specific PageRank: Iterate $R(t+1) = \beta\, M.R(t) + \left[\frac{1-\beta}{|S|}\right]_N$ until $|r(t+1) - r(t)| > \varepsilon$ where **S** is the set of pages relevant to the topic.
  - **R(t)** is the rank vector which store score after t iterations.
  - Here **ß** is the probability factor for not teleporting. This is used to handle **Spider traps**.

PROS AND CONS OF PAGERANK ALGORITHM:

PROS

- The algorithm is robust against Spam since its not easy for a webpage owner to add inlinks to his/her page from other important pages.
- PageRank is a global measure and is query independent.

CONS

- The major disadvantage of PageRank is that it favors the older pages, because a new page, even a very good one will not have many links unless it is a part of an existing site.

    - PageRank can be easily increased by the concept of ”link-farms” as shown below. However, while indexing, the search actively tries to find these flaws. To solve this problems we can use TrustRank.

BRIEF NOTES ON CODE EXECUTION:

Preprocessing:

- The dataset is read as a list by utils.py and then converted into a graph object using the NetworkX library. This does nothing but create a graph, and dead ends are not yet handled.
- The code is designed to deal with both directed and undirected graphs.

For PageRank:

- The pageRank.py code uses the structures and methods in the PageRank class to implement the algorithm.
    - It is first initialized with the graph produced by utils.py and other graph metadata.
    - Then the rank( ) method is executed on the class instance. This method iteratively computes the page rank for each node instead of maintaining an active matrix. <u>Hence it can handle sparse matrices as well.</u>
    - This method automatically handles <u>dead ends</u> as it sums all other values up to zero and the constant out link value remains. <u>Spider traps</u> are handled by the teleport.
- <u>NOTE ON USAGE FOR SPARSE MATRICES:</u>
  The central notion is that all data is always stored in the graph structure itself, instead of maintaining a matrix. The node ranks are stored in the nodes, and in each iteration, the node rank is updated by simple multiplication, instead of employing costly matrix multiplication.

RUNNING TIME

The code runs iteratively 15 times, and each iteration is linear in the number of nodes in the graph. (Thus it can handle sparse matrices.)

For the given dataset, on 1224 nodes, the algorithm takes 4 seconds.