COMPSCI 753

Algorithms for Massive Data

Assignment 3 / Semester 2, 2023

Graph Mining

General instructions and data

This assignment aims at exploring the PageRank algorithm on big real-world network data. By working on this assignment, you will learn how to implement some of the PageRank algorithms that we have learned in class.

Data: Download the Berkeley-Stanford web dataset 'web-BerkStan-final.txt' from the assignment page on Canvas[1]. Each line of the file represents a directed edge from a source node to a destination node. There are $N = 685230$ nodes. Nodes are represented by IDs ranging from 0 to 685229.

Submission

Please submit: (1) a file (.pdf or .html) that reports the answers requested for each task, and (2) a source code file (.py or .ipynb) that contains your code and detailed comments. Submit this on the Canvas assignment page by 23:59 NZST, Sunday 24 September. The files must contain your student ID, UPI and name.

Penalty Dates

The assignment will not be accepted after the last penalty date unless there are special circumstances (e.g., sickness with certificate). Penalties will be calculated as follows as a percentage of the marks for the assignment.

- **23:59 NZST, Sunday 24 September – No penalty**
- **23:59 NZST, Monday 25 September – 25% penalty**
- **23:59 NZST, Tuesday 26 September – 50% penalty**

[1]This dataset is adapted from SNAP http://snap.stanford.edu/data/web-BerkStan.html

Tasks (100 points)

Task 1 [40 points]: Implementation of Power Iteration Algorithm.

In this task you will implement the basic version of the Power Iteration algorithm for PageRank. This task involves two sub-tasks, as follows:

(A) [25 points] Implement the power iteration algorithm in matrix form to calculate the rank vector $\mathbf{r}$, without teleport, using the PageRank formulation:

$$\mathbf{r}^{(t+1)} = \mathbf{M} \cdot \mathbf{r}^{(t)}$$

The matrix $\mathbf{M}$ is an adjacency matrix representing nodes and edges from your downloaded dataset, with rows representing destination nodes and columns representing source nodes. This matrix is sparse[2]. Initialize $\mathbf{r}^{(0)} = [1/N, \ldots, 1/N]^T$. Let the stop criteria of your power iteration algorithm be $||\mathbf{r}^{(t+1)} - \mathbf{r}^{(t)}||_1 < 0.02$ (please note the stop criteria involves the L_1 norm). Spider traps and dead ends are not considered in this first task.

(B) [15 points] Run your code on the Berkeley-Stanford web data to calculate the rank score for all the nodes. Report: (1) The running time of your power iteration algorithm; (2) The number of iterations needed to stop; (3) The IDs and scores of the top-10 ranked nodes.

Task 2 [10 points]: Understanding dead-ends.

In this task, before extending your code to support dead-ends using teleport, you will run some analysis on your current implementation from Task 1. This second task involves two sub-tasks:

(A) [5 points] Calculate and report the number of dead-end nodes in your matrix $\mathbf{M}$.

(B) [5 points] Calculate the *leaked* PageRank score in each iteration of Task 1 (B). The *leaked* PageRank score is the total score you lose in that iteration because of dead-ends (hint: see example on slide 2 of W6.3 lecture notes). Create a plot that shows how this *leaked* score behaves as iterations progress. Explain the phenomenon you observe from this visualization.

[2]Consider using a sparse matrix (e.g., use scipy.sparse in Python) in your implementation, so that your algorithm should stop within a few seconds in a basic computer. If your algorithm can't stop within several minutes, you may want to check your implementation.

Task 3 [50 points]: Implementation of Power Iteration with Teleport.

In this task, you will extend your implementation from Task 1 using the teleport mechanism to handle both dead-ends and spider traps. This task involves three sub-tasks:

(A) [25 points] Extend your PageRank code to handle both spider traps and dead ends using the idea of teleport. In this task, your implementation will allow to teleport randomly to any node. Code the PageRank with teleport formulation that, using the sparse matrix $\mathbf{M}$, for each iteration works in three steps (slide 8 of W6.3 lecture notes):

Step 1: Calculate the $\mathbf{r}$ ranks of current iteration $\mathbf{r}^{new}$ (in matrix form):

$$\mathbf{r}^{new} = \beta \mathbf{M} \cdot \mathbf{r}^{old}$$

Step 2: Calculate the constant S for teleport:

$$S = \sum_j r_j^{new}$$

Step 3: Update $\mathbf{r}^{new}$ with teleport:

$$\mathbf{r}^{new} = \mathbf{r}^{new} + (1 - S)/N$$

In your implementation, use $\beta = 0.9$. Initialize $\mathbf{r}^{(0)} = [1/N, \ldots, 1/N]^T$. The stop criteria should be $||\mathbf{r}^{new} - \mathbf{r}^{old}||_1 < 0.02$.

(B) [15 points] Run your code on the Berkeley-Stanford web data to calculate the rank score for all the nodes. Report: (1) The running time; (2) The number of iterations needed to stop; (3) The IDs and scores of the top-10 ranked nodes.

(C) [10 points] Vary the teleport probability β with numbers in the set: $\{1, 0.9, 0.8, 0.7, 0.6\}$. Report the number of iterations needed to stop for each β. Explain, in words, your findings from this experiment.