General Instructions

Submission instructions: These questions require thought but do not require long answers. Please be as concise as possible. You should submit your answers as a writeup in PDF format via GradeScope and code via the Snap submission site.

Submitting writeup: Prepare answers to the homework questions into a single PDF file and submit it via http://gradescope.com. Make sure that the answer to each question is on a separate page. It is also important to tag your answers correctly on Gradescope. We will deduct 1 point for each incorrectly tagged subproblem. This means you can lose up to 10 points for incorrect tagging.

Submitting code: Upload your code at http://snap.stanford.edu/submit. Put all the code for a single question into a single file and upload it.

Questions

1 Dead ends in PageRank computations (20 points) [Wanzi, Sen, Jessica]

Let the matrix of the Web $M$ be an $n$-by-$n$ matrix, where $n$ is the number of Web pages. The entry $m_{ij}$ in row $i$ and column $j$ is 0, unless there is an arc from node (page) $j$ to node $i$. In that case, the value of $m_{ij}$ is $1/k$, where $k$ is the number of arcs (links) out of node $j$. Notice that if node $j$ has $k > 0$ arcs out, then column $j$ has $k$ values of $1/k$ and the rest 0’s. If node $j$ is a dead end (i.e., it has zero arcs out), then column $j$ is all 0’s.

Let $r = [r_1, r_2, \ldots, r_n]^T$ be (an estimate of) the PageRank vector; that is, $r_i$ is the estimate of the PageRank of node $i$. Define $w(r)$ to be the sum of the components of $r$; that is $w(r) = \sum_{i=1}^{n} r_i$.

In one iteration of the PageRank algorithm, we compute the next estimate $r'$ of the PageRank as: $r' = Mr$. Specifically, for each $i$ we compute $r'_i = \sum_{j=1}^{n} M_{ij} r_j$. 
(a) [6pts]

Suppose the Web has no dead ends. Prove that $w(r') = w(r)$.

(b) [7pts]

Suppose there are still no dead ends, but we use a teleportation probability of $1 - \beta$, where $0 < \beta < 1$. The expression for the next estimate of $r_i$ becomes $r'_i = \beta \sum_{j=1}^{n} M_{ij} r_j + (1 - \beta)/n$. Under what circumstances will $w(r') = w(r)$? Prove your conclusion.

(c) [7pts]

Now, let us assume a teleportation probability of $1 - \beta$ in addition to the fact that there are one or more dead ends. Call a node “dead” if it is a dead end and “live” if not. Assume $w(r) = 1$. At each iteration, each live node $j$ distributes $(1 - \beta)r_j/n$ PageRank to each of the other nodes, and each dead node $j$ distributes $r_j/n$ PageRank to each of the other nodes.

Write the equation for $r'_i$ in terms of $\beta$, $M$, $r$, $n$, and $D$ (where $D$ is the set of dead nodes). Then, prove that $w(r')$ is also 1.

What to submit

(i) Proof [1(a)]

(ii) Condition for $w(r') = w(r)$ and Proof [1(b)]

(iii) Equation for $r'_i$ and Proof [1(c)]

2 Implementing PageRank and HITS (25 points) [Yu-tian, Kush, Chang]

In this problem, you will learn how to implement the PageRank and HITS algorithms in Spark. You will be experimenting with a small randomly generated graph (assume graph has no dead-ends) provided at http://snap.stanford.edu/class/cs246-data/graph-full.txt.

It has $n = 1000$ nodes (numbered $1, 2, \ldots, 1000$), and $m = 8192$ edges, 1000 of which form a directed cycle (through all the nodes) which ensures that the graph is connected. It is easy to see that the existence of such a cycle ensures that there are no dead ends in the graph. There may be multiple directed edges between a pair of nodes, and your solution should treat them as the same edge. The first column in graph-full.txt refers to the source node, and the second column refers to the destination node.
(a) PageRank Implementation [12 points]

Assume the directed graph $G = (V, E)$ has $n$ nodes (numbered 1, 2, ..., $n$) and $m$ edges, all nodes have positive out-degree, and $M = [M_{ji}]_{n \times n}$ is a an $n \times n$ matrix as defined in class such that for any $i, j \in [1, n]$:

$$M_{ji} = \begin{cases} \frac{1}{\text{deg}(i)} & \text{if } (i \rightarrow j) \in E, \\ 0 & \text{otherwise.} \end{cases}$$

Here, $\text{deg}(i)$ is the number of outgoing edges of node $i$ in $G$. If there are multiple edges in the same direction between two nodes, treat them as a single edge. By the definition of PageRank, assuming $1 - \beta$ to be the teleport probability, and denoting the PageRank vector by the column vector $r$, we have the following equation:

$$r = \frac{1 - \beta}{n} \mathbf{1} + \beta Mr,$$

(1)

where $\mathbf{1}$ is the $n \times 1$ vector with all entries equal to 1.

Based on this equation, the iterative procedure to compute PageRank works as follows:

1. Initialize: $r^{(0)} = \frac{1}{n} \mathbf{1}$

2. For $i$ from 1 to $k$, iterate: $r^{(i)} = \frac{1 - \beta}{n} \mathbf{1} + \beta M r^{(i-1)}$

Run the aforementioned iterative process in Spark for 40 iterations (assuming $\beta = 0.8$) and obtain the PageRank vector $r$. In particular, you don’t have to implement the blocking algorithm from lecture. The matrix $M$ can be large and should be processed as an RDD in your solution. Compute the following:

- List the top 5 node ids with the highest PageRank scores.
- List the bottom 5 node ids with the lowest PageRank scores.

For a sanity check, we have provided a smaller dataset http://snap.stanford.edu/class/cs246-data/graph-small.txt. In that dataset, the top node has id 53 with value 0.036.

(b) HITS Implementation [13 points]

Assume the directed graph $G = (V, E)$ has $n$ nodes (numbered 1, 2, ..., $n$) and $m$ edges, all nodes have non-negative out-degree, and $L = [L_{ij}]_{n \times n}$ is a an $n \times n$ matrix referred to as the link matrix such that for any $i, j \in [1, n]$:

$$L_{ij} = \begin{cases} 1 & \text{if } (i \rightarrow j) \in E, \\ 0 & \text{otherwise.} \end{cases}$$

Given the link matrix $L$ and some scaling factors $\lambda, \mu$, the hubbiness vector $h$ and the authority vector $a$ can be expressed using the equations:

$$h = \lambda L a, \quad a = \mu L^T h$$

(2)

where $1$ is the $n \times 1$ vector with all entries equal to 1.

Based on this equation, the iterative method to compute $h$ and $a$ is as follows:

1. Initialize $h$ with a column vector (of size $n \times 1$) of all 1’s.
2. Compute $a = L^T h$ and scale so that the largest value in the vector $a$ has value 1.
3. Compute $h = La$ and scale so that the largest value in the vector $h$ has value 1.
4. Go to step 2.

Repeat the iterative process for 40 iterations, assume that $\lambda = 1, \mu = 1$ and then obtain the hubbiness and authority scores of all the nodes (pages). The link matrix $L$ can be large and should be processed as an RDD. Compute the following:

- List the 5 node ids with the highest hubbiness score.
- List the 5 node ids with the lowest hubbiness score.
- List the 5 node ids with the highest authority score.
- List the 5 node ids with the lowest authority score.

For a sanity check, you should confirm that `graph-small.txt` has highest hubbiness node id 59 with value 1 and highest authority node id 66 with value 1.

**What to submit**

(i) List 5 node ids with the highest and least PageRank scores [2(a)]
(ii) List 5 node ids with the highest and least hubbiness and authority scores [2(b)]
(iii) Upload all the code to the snap submission site [2(a) & 2(b)]
3  Clique-Based Communities (25 points) [Praty, Dylan, Hiroto]

Imagine an undirected graph $G$ with nodes 2, 3, 4, ..., 1000000. (Note that there is no node 1.) There is an edge between nodes $i$ and $j$ if and only if $i$ and $j$ have a common factor other than 1. Put another way, the only edges that are missing are those between nodes that are relatively prime; e.g., there is no edge between 15 and 56.

We want to find communities by starting with a clique (not a bi-clique) and growing it by adding nodes. However, when we grow a clique, we want to keep the density of edges at 1; i.e., the set of nodes remains a clique at all times. A maximal clique is a clique for which it is impossible to add a node and still retain the property of being a clique; i.e., a clique $C$ is maximal if every node not in $C$ is missing an edge to at least one member of $C$.

(a) [5 points]

Prove that if $i$ is any integer greater than 1, then the set $C_i$ of nodes of $G$ that are divisible by $i$ is a clique.

(b) [10 points]

Under what circumstances is $C_i$ a maximal clique? Prove that your conditions are both necessary and sufficient. (Trivial conditions, like “$C_i$ is a maximal clique if and only if $C_i$ is a maximal clique,” will receive no credit.)

(c) [10 points]

Prove that $C_2$ is the unique largest clique. That is, it has more elements than any other clique.

What to submit

(i) Proof that the specified nodes are a clique.

(ii) Necessary and sufficient conditions for $C_i$ to be a maximal clique, with proof.

(iii) Proof that $C_2$ is the unique largest clique.
4 Dense Communities in Networks (20 points) [Sanyam, Qijia, Heather]

In this problem, we study the problem of finding dense communities in networks.

Definitions: Assume $G = (V, E)$ is an undirected graph (e.g., representing a social network).

- For any subset $S \subseteq V$, we let the induced edge set (denoted by $E[S]$) to be the set of edges both of whose endpoints belong to $S$.
- For any $v \in S$, we let $\text{deg}_S(v) = |\{u \in S | (u, v) \in E\}|$.
- Then, we define the density of $S$ to be:
  $$\rho(S) = \frac{|E[S]|}{|S|}.$$ 
- Finally, the maximum density of the graph $G$ is the density of the densest induced subgraph of $G$, defined as:
  $$\rho^*(G) = \max_{S \subseteq V} \{\rho(S)\}.$$

Goal. Our goal is to find an induced subgraph of $G$ whose density is not much smaller than $\rho^*(G)$. Such a set is very densely connected, and hence may indicate a community in the network represented by $G$. Also, since the graphs of interest are usually very large in practice, we would like the algorithm to be highly scalable. We consider the following algorithm:

Require: $G = (V, E)$ and $\epsilon > 0$

$\tilde{S}, S \leftarrow V$

while $S \neq \emptyset$ do

\[ A(S) := \{i \in S | \text{deg}_S(i) \leq 2(1 + \epsilon)\rho(S)\}\]

$S \leftarrow S \setminus A(S)$

if $\rho(S) > \rho(\tilde{S})$ then

$\tilde{S} \leftarrow S$

end if

end while

return $\tilde{S}$

The basic idea in the algorithm is that the nodes with low degrees do not contribute much to the density of a dense subgraph, hence they can be removed without significantly influencing the density.

We analyze the quality and performance of this algorithm. We start with analyzing its performance.
(a) [10 points]

We show through the following steps that the algorithm terminates in a logarithmic number of steps.

i. Prove that at any iteration of the algorithm, $|A(S)| \geq \frac{\epsilon}{1+\epsilon}|S|$.

ii. Prove that the algorithm terminates in $O(\log_{1+\epsilon}(n))$ iterations, where $n$ is the initial number of nodes.

(b) [10 points]

We show through the following steps that the density of the set returned by the algorithm is at most a factor $2(1+\epsilon)$ smaller than $\rho^*(G)$.

i. Assume $S^*$ is the densest subgraph of $G$. Prove that for any $v \in S^*$, we have: $\text{deg}_{S^*}(v) \geq \rho^*(G)$.

ii. Consider the first iteration of the while loop in which there exists a node $v \in S^* \cap A(S)$. Prove that $2(1+\epsilon)\rho(S) \geq \rho^*(G)$.

iii. Conclude that $\rho(\tilde{S}) \geq \frac{1}{2(1+\epsilon)}\rho^*(G)$.

What to submit

(a)  
i. Proof of $|A(S)| \geq \frac{\epsilon}{1+\epsilon}|S|$.
   ii. Proof of number of iterations for algorithm to terminate.

(b)  
i. Proof of $\text{deg}_{S^*}(v) \geq \rho^*(G)$.
   ii. Proof of $2(1+\epsilon)\rho(S) \geq \rho^*(G)$.
   iii. Conclude that $\rho(\tilde{S}) \geq \frac{1}{2(1+\epsilon)}\rho^*(G)$. 