CS246: Mining Massive Data Sets **Winter 2020**

Problem Set 3

Please read the homework submission policies at http://cs246.stanford.edu.

1 Dead ends in PageRank computations (25 points)

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

In one iteration of the PageRank algorithm, we compute the next estimate $\mathbf{r}'$ of the PageRank as: $\mathbf{r}' = M\mathbf{r}$. Specifically, for each i we compute $r'_i = \sum_{j=1}^{n} M_{ij} r_j$. Define $w(\mathbf{r}')$ to be the sum of components of r'; that is $w(\mathbf{r}') = \sum_{i=1}^{n} r'_i$.

You may use D (the set of dead nodes) in your equation.

(a) [6pts]

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

★ **SOLUTION:** We need to show $\sum_{i=1}^{n} \sum_{j=1}^{n} M_{ij} r_j = \sum_{i=1}^{n} r_i$. Interchange order of summations so we have $\sum_{j=1}^{n} (\sum_{i=1}^{n} M_{ij}) r_j$ on the left. Since there are no dead ends, $\sum_{i=1}^{n} M_{ij} = 1$ for each j. Thus the equation holds.

(b) [9pts]

Suppose there are still no dead ends, but we use a teleportation probability of $1-\beta$ where we teleport to a random node ($0 < \beta < 1$). 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 $w(\mathbf{r}') = w(\mathbf{r})$? Prove your conclusion.

★ **SOLUTION:** If and only if $w(\mathbf{r}) = 1$. To see why, sum over I to get $w(\mathbf{r}') = \sum_{i=1}^{n} \beta \sum_{j=1}^{n} M_{ij} r_j + \sum_{i=1}^{n} (1-\beta)/n$. The second term on the right sums to $1-\beta$, and using the reasoning from part (a), the first term on the right sums to $\beta \sum_{j=1}^{n} r_j$. That is: $w(\mathbf{r}') = \beta w(\mathbf{r}) + 1 - \beta$. If $w(\mathbf{r}) = 1$, then the equation tells us $w(\mathbf{r}')$ is also 1. Conversely, if $w(\mathbf{r}') = w(\mathbf{r}) = x$, then x must satisfy the equation $x = \beta x + 1 - \beta$, from which it follows that $x = 1$.

(c) [10pts]

Now let us assume there are one or more dead ends. Call a node "dead" if it is a dead end and "live" if not. At each iteration, we teleport from live nodes with probability $1-\beta$ and teleport from dead nodes with probability 1. In both cases, we choose a random node uniformly to teleport to. Assume $w(\mathbf{r}) = 1$.

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

★ **SOLUTION:** The equation is $r'_i = \beta \sum_{j=1}^{n} M_{ij} r_j + (1-\beta)/n + (\beta/n) \sum_{\text{dead } j} r_j$ If we sum over i and use the same trick of moving the summation on i to apply only to M_{ij}, we get $w(\mathbf{r}') = \beta \sum_{\text{live} j} r_j + 1 - \beta + \beta \sum_{\text{dead} j} r_j$. The first and last terms on the right together give $\beta w(\mathbf{r})$, which is β, since $w(\mathbf{r})$ is assumed to be 1. Thus, the right side reduces to 1, as desired.

What to submit

(i) Proof [1(a)]

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

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

2 Implementing PageRank and HITS (30 points)

In this problem, you will learn how to implement the PageRank and HITS algorithms in Spark. The general computation should be done in Spark, and you may also include numpy operations whenever needed. You will be experimenting with a small randomly generated graph (assume graph has no dead-ends) provided at `graph-full.txt`.

There are 100 nodes ($n = 100$) in the small graph and 1000 nodes ($n = 1000$) in the full graph, 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.

Implementation hint: You may choose to store the PageRank vector $\mathbf{r}$ either in memory or as an RDD. Only the matrix M of links is too large to store in memory, and you are allowed to store matrix M in an RDD. e.g. data = sc.textFile("graph−full.txt"). On an actual cluster, an RDD is partitioned across the nodes of the cluster. However, you cannot then M = data.collect() which fetches the entire RDD to a single machine at the driver node and stores it as an array locally.

(a) PageRank Implementation [15 points]

Assume the directed graph $G = (V, E)$ has n nodes (numbered $1, 2, \ldots, n$) 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 $i, j \in [\![1, n]\!]$:

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

Here, $\deg(i)$ 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:

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

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

1. Initialize: $\mathbf{r}^{(0)} = \frac{1}{n}\mathbf{1}$
2. For i from 1 to k, iterate: $\mathbf{r}^{(i)} = \frac{1-\beta}{n}\mathbf{1} + \beta M \mathbf{r}^{(i-1)}$

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 PageRank scores and report the node id for the following using `graph-full.txt`:

- 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 (`graph-small.txt`). In that dataset, the top node has id 53 with value 0.036. Note that the `graph-small.txt` dataset is only provided for sanity check purpose. Your write-up should include results obtained using `graph-full.txt` (for both part (a) and (b)).

★ **SOLUTION:** Top 1st node id = 263
Top 2nd node id = 537
Top 3rd node id = 965
Top 4th node id = 243
Top 5th node id = 285

Bottom 1st node id = 558
Bottom 2nd node id = 93
Bottom 3rd node id = 62
Bottom 4th node id = 424
Bottom 5th node id = 408

(b) HITS Implementation [15 points]

Assume the directed graph $G = (V, E)$ has n nodes (numbered $1, 2, \ldots, n$) 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 $i, j \in [\![1, n]\!]$:

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

Given the link matrix L and some scaling factors λ, μ, the hubbiness vector h and the authority vector a can be expressed using the equations:

$$h = \lambda L a, a = \mu L^T h \tag{2}$$

where $\mathbf{1}$ 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 using `graph-full.txt`:

- 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.

★ **SOLUTION:** Top 1th hubbiness node id = 840
Top 2th hubbiness node id = 155
Top 3th hubbiness node id = 234
Top 4th hubbiness node id = 389
Top 5th hubbiness node id = 472

Bottom 1th hubbiness node id = 23
Bottom 2th hubbiness node id = 835
Bottom 3th hubbiness node id = 141
Bottom 4th hubbiness node id = 539
Bottom 5th hubbiness node id = 889

Top 1th authority node id = 893
Top 2th authority node id = 16
Top 3th authority node id = 799
Top 4th authority node id = 146
Top 5th authority node id = 473

Bottom 1th authority node id = 19
Bottom 2th authority node id = 135
Bottom 3th authority node id = 462
Bottom 4th authority node id = 24
Bottom 5th authority node id = 910

What to submit

(i) List 5 node ids with the highest and least PageRank scores [2(a)] using `graph-full.txt`

(ii) List 5 node ids with the highest and least hubbiness and authority scores [2(b)] using `graph-full.txt`

(iii) Upload all the code via Gradescope [2(a) & 2(b)]

3 Clique-Based Communities (25 points)

Imagine an undirected graph G with nodes $2, 3, 4, \ldots, 1000000$. (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.

Let C_i be the set of nodes of G that are divisible by i, where i is a positive integer.

(a) [5 points]

Prove that C_i is a clique for any i.

★ **SOLUTION:** If there are fewer than two nodes in C_i, then C_i is a clique by definition. Otherwise, any two nodes in C_i have i as a common factor and therefore have an edge between them.

(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.)

★ **SOLUTION:** i must be a prime less than or equal to one million.

If i is greater than one million, then C_i is the empty clique, and adding any node to C_i will produce a 1-clique. Thus C_i is not maximal.

If i is less than or equal to one million, but isn't prime, let j be a factor of i, with $1 < j < i$. Node j is not in C_i, yet it has an edge to every member of C_i, because it has j as a common factor. Thus, C_i is not maximal.

Conversely, if i is prime, and less than or equal to one million, then there is no node outside C_i that has an edge to the node i itself. To see why, suppose j were such a node. Then i and j have a common factor other than 1, which can only be i, since i is prime. However, if j has i as a factor, then j is a multiple of i and therefore already in C_i.

(c) [10 points]

Prove that C_2 is the unique largest clique. That is, it has more elements than any other clique. (**Note**: Not all cliques are in the form of C_i)

★ **SOLUTION:** i and $i+1$ are always relatively prime. This is because if they had a common factor $p > 1$, then $(i+1) - i = 1$ must also be divisible by p, which is impossible for any $p > 1$. Since i and $i+1$ are always relatively prime, only one of $\{2,3\}, \{4,5\}, \{6,7\}, \ldots$ can be in a clique. Therefore, the largest possible clique has $500,000$ members, as does C_2.

Conversely, there are no cliques other than C_2 that have $500,000$ members. Any clique with $500,000$ members must contain either 2 or 3, because otherwise (by the previous argument) the clique must contain at most $499,999$ members. If it contains 3, then all other elements in the clique must have 3 as a factor (since 3 is prime), and there are fewer than $499,999$ elements that are divisible by 3.

Note that any solution that implicitly assumes that all cliques are of the form C_i is incorrect. As a simple example, consider the clique $(2*3, 2*5, 3*5)$; there is an edge between any two of the nodes, but there is no number that divides all three of the nodes. An example of an incorrect answer is "by part b, all maximal cliques must be of the form C_i, where i is prime; since C_2 is larger than C_3, C_5, etc., it is the unique largest 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)

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 $\deg_S(v) = |\{u \in S | (u, v) \in E\}|$.

- 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**
    $A(S) := \{i \in S \mid \deg_S(i) \leq 2(1+\epsilon)\rho(S)\}$
    $S \leftarrow S \setminus A(S)$
    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.

★ **SOLUTION:** We have: $2|E[S]| = \sum_{v \in S} \deg_S(v) \geq \sum_{v \in S \setminus A(S)} \deg_S(v) > |S \setminus A(S)| 2(1+\epsilon)\rho(S)$. Hence, $|S \setminus A(S)| < |S|/(1+\epsilon)$, or equivalently $|A(S)| > \frac{\epsilon}{1+\epsilon}|S|$. At each iteration, the size of S goes down by a factor larger than $1+\epsilon$. Starting with n nodes, in at most $O(\log_{1+\epsilon}(n))$ iterations the set gets depleted.

(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: $\deg_{S^*}(v) \geq \rho^*(G)$.

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

★ **SOLUTION:** If for some $v \in S^*$, we have $\deg^*_S(v) < \rho^*(G)$, then the set $S^* \setminus \{v\}$ has density $\frac{|E[S^*]| - \deg_{S^*}}{|S^*|-1} > \rho(S^*)$, which contradicts the assumption of S^* being the densest subgraph. If there is a first iteration where there exists a node $v \in S^* \cap A(S)$, we have: $2(1+\epsilon)\rho(S) \geq \deg_{S^*}(v) \geq \rho^*(G)$, hence $2(1+\epsilon)\rho(S) \geq \rho^*(G)$. Finally, notice that since we proved in the last part of the problem that the set S eventually gets depleted, there must be an iteration in which nodes from S^* start to get removed from S. We proved at that iteration $\rho(S) \geq \frac{1}{2(1+\epsilon)}\rho^*(G)$. Noticing $\rho(\tilde{S}) \geq \rho(S)$ finishes the proof.

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