**CSE 547: Machine Learning for Big Data** **Spring 2019**

# Problem Set 3

Please read the homework submission policies.

**Assignment Submission** All students should submit their assignments electronically via GradeScope. Students may typeset or scan their **neatly written** homeworks (points **will** be deducted for illegible submissions). Simply sign up on Gradescope and use the course code 97EWEW. Please use your UW NetID if possible.

For the non-coding component of the homework, you should upload a PDF rather than submitting as images. We will use Gradescope for the submission of code as well. Please make sure to tag each part correctly on Gradescope so it is easier for us to grade. There will be a small point deduction for each mistagged page and for each question that includes code. Put all the code for a single question into a single file and upload it. Only files in text format (e.g. .txt, .py, .java) will be accepted. **There will be no credit for coding questions without submitted code on Gradescope, or for submitting it after the deadline**, so please remember to submit your code.

**Late Day Policy** All students will be given two no-questions-asked late periods, but only one late period can be used per homework and cannot be used for project deliverables. A late-period lasts 48 hours from the original deadline (so if an assignment is due on Thursday at 11:59 pm, the late period goes to the Saturday at 11:59pm Pacific Time).

**Academic Integrity** We take academic integrity extremely seriously. We strongly encourage students to form study groups. Students may discuss and work on homework problems in groups. However, each student must write down the solutions and the code independently. In addition, each student should write down the set of people whom they interacted with.

Discussion Group (People with whom you discussed ideas used in your answers):

On-line or hardcopy documents used as part of your answers:

I acknowledge and accept the Academic Integrity clause.

*(Signed)*_____

# 1 Dead Ends in PageRank Computations (25 points)

Suppose we denote the matrix of the Internet as the $n$-by-$n$ matrix $M$, where $n$ is the number of webpages. Suppose there are $k$ links out of the node (webpage) $j$, and

$$M_{ij} = \begin{cases} 1/k & \text{if there is a link from } j \text{ to } i \\ 0 & \text{otherwise} \end{cases}$$

For a webpage $j$ that is a *dead end* (i.e., one having zero links out), the column $j$ is all zeroes.

Let $\mathbf{r} = [r_1, r_2, \ldots, r_n]^\top$ be *an estimate* of the PageRank vector. In one iteration of the PageRank algorithm, we compute the next estimate $\mathbf{r'}$ of the PageRank as: $\mathbf{r'} = M\mathbf{r}$.

Given any PageRank estimate vector $\mathbf{r}$, define $w(\mathbf{r}) = \sum_{i=1}^{n} r_i$.

(a) **[6pts]** Suppose the Web has no dead ends. Prove that $w(\mathbf{r'}) = w(\mathbf{r})$.

(b) **[9pts]** 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 $w(\mathbf{r'}) = w(\mathbf{r})$? Prove your conclusion.

(c) **[10pts]** 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 $w(\mathbf{r}) = 1$. At each iteration, when not teleporting, each live node $j$ distributes $\beta r_j$ PageRank uniformly across each of the nodes it links to, and each dead node $j$ distributes $r_j/n$ PageRank to all the nodes.

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

## 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. You will be experimenting with a small randomly generated graph (assume that the graph has no dead-ends) provided in `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 $r$ either in memory or as an RDD. Only the matrix of links is too large to store in memory.*

### (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 $i, j \in \{1, \ldots, n\}$:

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

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:

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

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 estimate of) the PageRank vector $r$. 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 (`graph-small.txt`). In that dataset, the top node has id 53 with value 0.036.

### (b) HITS Implementation [15 points]

Assume that 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 $i, j \in \{1, \ldots, n\}$:

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

Given the link matrix $L$ and scaling factors $\lambda$ and $\mu$, the hubbiness vector $h$ (from the word "hub") and the authority vector $a$ can be expressed using the equations:

$$h = \lambda L a \tag{2}$$

$$a = \mu L^T h \tag{3}$$

Assume that $\lambda = 1$, $\mu = 1$. Then 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, and 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 Gradescope [2(a) & 2(b)]

# 3 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:

**Algorithm 1**

**Require:** $G = (V, E)$ and $\epsilon > 0$
1: $\widetilde{S}, S \leftarrow V$
2: **while** $S \neq \emptyset$ **do**
3: $\quad A(S) := \{i \in S \mid \deg_S(i) \leq 2(1 + \epsilon)\rho(S)\}$
4: $\quad S \leftarrow S \setminus A(S)$
5: $\quad$ **if** $\rho(S) > \rho(\widetilde{S})$ **then**
6: $\quad\quad \widetilde{S} \leftarrow S$
7: $\quad$ **end if**
8: **end while**
9: **return** $\widetilde{S}$

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.

### (a) Performance Guarantee [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) Quality Guarantee [10 points]

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

i. Assume that $S^*$ 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 $\rho(\widetilde{S}) \geq \frac{1}{2(1+\epsilon)}\rho^*(G)$.

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

# 4 Spectral Clustering (25 points)

We saw in lecture several methods for partitioning different types of graphs. In this problem, we explore (yet another) such partitioning method called "spectral clustering". The name derives from the fact that it uses the *spectrum of certain matrices* (that is, the eigenvalues and eigenvectors) derived from the graph.

Our overarching goals in this problem are to (1) derive a simple algorithm for spectral clustering, and (2) see how it could also be used in finding a clustering that maximizes

modularity, something for which we already saw an algorithm in class (Louvain's algorithm).

Let us first fix the notation we'll use in this problem.

- Let $G = (V, E)$ be a simple (that is, no self- or multi-edges), undirected, connected graph with $n = |V|$ and $m = |E|$.
- We use the notation $\{i, j\} \in E$ to denote that the nodes $i$ and $j$ are connected via an edge (note that since this is an undirected graph, we do not talk about the direction of the connection).
- Let $A$ be the adjacency matrix of $G$: that is, $A_{ij} = \begin{cases} 1 & \text{if } \{i,j\} \in E \\ 0 & \text{otherwise} \end{cases}$.
- We use $d_i$ to denote the degree of the $i$-th node; by definition of the adjacency matrix, $d_i = \sum_{j=1}^{n} A_{ij}$. We define the diagonal matrix $D$ formed by placing the degrees of the nodes along its diagonal. That is, $D_{ii} = d_i$ for all $i = 1, 2, \ldots, n$.
- We define the graph Laplacian as the $n \times n$ matrix $L = D - A$.
- We define a vector $e_i \in \mathbb{R}^n$ as zero on all coordinates *except* the $i$-th, at which it is 1. In this case, since $|V| = n$, the vector $e_i$ is $n$-dimensional.
- Define the vector $e \in \mathbb{R}^n$ as the vector of all 1s. Again, $e$ is an $n$-dimensional vector in this case.

For a set of nodes $S \subseteq V$, we associate two values that measure, in some sense, its quality as a cluster: the "cut" and the "volume". We define these two values below.

The "cut" of a set $S$ is defined as the number of edges that have one end point in the set $S$ and the other in its complement, $\overline{S} = V \backslash S$:

$$\text{cut}(S) = \sum_{i \in S, j \in \overline{S}} A_{ij}. \tag{4}$$

Observe that by definition, $\text{cut}(S) = \text{cut}(\overline{S})$. The "volume" of a set is defined as the sum of degrees of nodes in $S$:

$$\text{vol}(S) = \sum_{i \in S} d_i, \tag{5}$$

where $d_i$ is the degree of node $i$.

In addition to the above measures associated with set $S$, we define the *normalized cut* of a graph (associated with a partitioning $S$) as

$$\text{NCUT}(S) = \frac{\text{cut}(S)}{\text{vol}(S)} + \frac{\text{cut}(\overline{S})}{\text{vol}(\overline{S})} \tag{6}$$

For a set $S$ to have a small normalized cut value, it must have very few edges connecting the nodes inside $S$ to the rest of the graph (making the numerators small), *as well as* roughly equal volumes of $S$ and $\overline{S}$, so that neither denominator is too small.

We are now ready to start proving things. **Please be careful when reading expressions involving $S$ and $\overline{S}$, since at a quick glance they may look the same.**

### (a) Establishing Some Basics [10 points]

We first make some observations that will help us formulate the problem of minimizing normalized cut nicely in the next sub-problem.

Given a set of nodes $S$, we define a vector $x_S \in \mathbb{R}^n$, such that the $i$-th coordinate $x_S^{(i)}$ of $x_S$ is defined as follows:

$$x_S^{(i)} = \begin{cases} \sqrt{\frac{\text{vol}(\overline{S})}{\text{vol}(S)}} & \text{if } i \in S \\ -\sqrt{\frac{\text{vol}(S)}{\text{vol}(\overline{S})}} & \textit{otherwise} \end{cases} \tag{7}$$

**To clarify (because the font may not be clear), in Equation 7, in the case $i \in \overline{S}$, the term in the denominator under the square root is vol$(\overline{S})$.**

In the following, we are using the notation established at the start of this problem and some set of node $S \subseteq V$. Prove the following statements:

1. $L = \sum_{\{i,j\} \in E} (e_i - e_j)(e_i - e_j)^\top$.
2. For any vector $x \in \mathbb{R}^n$, it holds that $x^\top L x = \sum_{\{i,j\} \in E} (x_i - x_j)^2$.
3. $x_S^\top L x_S = c \cdot \text{NCUT}(S)$ for some constant $c$ that depends on the problem parameters. Note that you should specify this constant.
4. $x_S^\top D e = 0$.
5. $x_S^\top D x_S = 2m$.

### (b) Normalized Cut Minimization [8 points]

Based on the facts we just proved about $x$ chosen as per Equation 7, we can formulate the normalized cut minimization problem as follows:

$$\begin{array}{ll} \underset{S \subset V}{\text{minimize}} & \frac{x_S^\top L x_S}{x_S^\top D x_S} \\ \text{subject to} & x_S^\top D e = 0, \\ & x_S^\top D x_S = 2m. \end{array}$$

To be clear, we are minimizing over all (non-trivial) partitions $(S, \overline{S})$, where the vectors $x_S$ are defined as described in Equation 7. Note that the two constraints appearing in the

optimization are trivially maintained due to the form $x_S$ as we have shown in the previous sub-problem. However, constraining $x$ to takes the form of Equation 7 makes this optimization problem NP-Hard. We will instead relax the optimization problem, making it tractable, and then round the relaxed solution back to a feasible point of the *original* problem. The relaxation we choose eliminates the constraint on the form of $x$. This gives us the following *relaxed* optimization problem:

$$\begin{array}{ll} \underset{x\in\mathbb{R}^n}{\text{minimize}} & \frac{x^\top Lx}{x^\top Dx}, \\ \text{subject to} & x^\top De = 0, \\ & x^\top Dx = 2m. \end{array} \tag{8}$$

Show that the minimizer of the optimization problem 8 is

$$x^* = D^{-1/2}v,$$

where $v$ is an eigenvector corresponding to the second smallest eigenvalue of the *normalized graph Laplacian* $\mathcal{L} = D^{-1/2}LD^{-1/2}$.

Finally, to round the solution back to a feasible point in the original problem, we can take the nodes corresponding to the positive entries of the eigenvector to be in the set $S$ and those corresponding to the negative entries to be in $\overline{S}$.

*Hint 1: Use the linear transformation $z = D^{1/2}x$.*

*Hint 2: Prove that $e$ is the eigenvector corresponding to the smallest eigenvalue of $L$, and use this fact.*

*Hint 3: For a symmetric matrix, we can always find eigenvectors that form an orthogonal basis for $\mathbb{R}^n$.*

### (c) Relating Modularity to Cuts and Volumes [7 points]

In class, we presented the modularity of a graph clustering in the context of the Louvain Algorithm. Modularity actually relates to cuts and volumes as well. Let us consider a partitioning of our graph $A$ into two clusters, and let $y \in \{1, -1\}^n$ be an assignment vector for a set $S$:

$$y_i = \begin{cases} 1 & \text{if } i \in S \\ -1 & \text{otherwise} \end{cases} \tag{9}$$

Then, the *modularity* of the assignment $y$ is

$$Q(y) = \frac{1}{2m}\sum_{i,j=1}^{n}\left[A_{ij} - \frac{d_i d_j}{2m}\right]\delta(y_i, y_j). \tag{10}$$

Here $\delta(y_i, y_j)$ is an indicator for whether or not the nodes $i$ and $j$ are in the same cluster:

$$\delta(y_i, y_j) = \begin{cases} 1 & \text{if } i \text{ and } j \text{ are both in the same cluster} \\ 0 & \text{otherwise} \end{cases}$$

Let $y$ be the assignment vector in Equation 9. Prove that

$$Q(y) = \frac{1}{2m}\left(-2 \cdot \text{cut}(S) + \frac{1}{m}\text{vol}(S) \cdot \text{vol}(\overline{S})\right) \tag{11}$$

Thus, maximizing modularity is really just minimizing the sum of the cut and the negative product of the partition's volumes. As a result, we can use spectral algorithms similar to the one derived in parts 1-2 in order to find a clustering that maximizes modularity. While this might provide an intuitively "better" clustering after inspection than the Louvain Algorithm, spectral algorithms are computationally intensive on large graphs, and would only partition the graph into 2 clusters.

*Note: You only need to prove the relationship between modularity and cuts; you do* ***not*** *need to derive the actual spectral algorithm.*

## What to submit

i. Proof of of the 5 equalities in part 4(a)

ii. Proof that the minimizer of the optimization problem 8 is $x^* = D^{-1/2}v$ [4(b)]

iii. Proof of Equation 11 [4(c)]