CSE 547: Machine Learning for Big Data **Spring 2020**

Problem Set 3

Please read the homework submission policies.

Assignment Submission All students should submit their assignments electronically via GradeScope. No handwritten work will be accepted. Math formulas **must** be typeset using LaTeX or other word processing software that supports mathematical symbols (E.g. Google Docs, Microsoft Word). Simply sign up on Gradescope and use the course code MP8KGN. 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.

Coding You may use any programming languages and standard libraries, such as NumPy and PySpark, but you may not use specialized packages and, in particular, machine learning libraries (e.g. sklearn, TensorFlow), unless stated otherwise. Ask on the discussion board whether specific libraries are allowed if you are unsure.

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 βr_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 β, 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 The Louvain Algorithm for Community Detection (35 points)

Note: For this question, assume all graphs are undirected and weighted.

Communities, or clusters of densely linked nodes in a graph, are important elements of a graph's structure. Thus, discovering communities from an observed graph can help us summarize the overall structure of a graph and learn more about the underlying process. However, finding the "best" set of communities from data is often a difficult problem. One way to measure how well a network is partitioned into communities is to calculate the number of within-community edges relative to the number of between-community edges. This can be formalized using *modularity*, defined as:

$$Q = \frac{1}{2m} \sum_{1 \leq i,j \leq n} \left(\left[A_{ij} - \frac{d_i d_j}{2m} \right] \delta\left(c_i, c_j\right) \right)$$

Where A is the adjacency matrix of a graph G with n vertices and m edges, A_{ij} is the (i, j)-th entry of A, $2m = \sum_{i,j} A_{ij}$ is the sum of all entries in A, d_i is the degree of node i, δ is the Kronecker delta, i.e. $\delta(k, l) = 1$ when $k = l$, otherwise - $\delta(k, l) = 0$, and c_i and c_j are the communities of i and j respectively. Here, we assume that communities are disjoint, i.e. each node can only belong to one community. The modularity of a graph lies in the range $[-1, 1]$.

Maximizing the modularity of a given graph is a computationally hard problem. The Louvain algorithm is a popular and efficient heuristic used to solve this problem. It is a greedy algorithm, meaning that at every step, it will take the path that provides the largest possible increase to the objective (modularity) at that step. Each pass of the algorithm has two phases:

- **Phase 1 (Modularity Optimization)** aims to group nodes in the graph G into communities in a way that maximizes the modularity of the graph. After the first pass, the input graph to this step is the graph produced by phase 2 in the previous pass (see below).
- **Phase 2 (Community Aggregation)** combines each community into a single node, producing a new graph H where each node represents a community of nodes in the graph G. This new graph H is fed into phase 1 in the next pass of the algorithm.

We repeat these two phases until we no longer increase modularity through one pass. The algorithm proceeds as follows:

Phase 1 (Modularity Optimization)

Input: a graph $G = (V, E)$ with a vertex set V and an edge set E. ▷ The input changes in each pass of the algorithm; after pass 1 the input is the graph output by phase 2.

Output: a partition of G into communities.

Initialize each node i as its own community
for each node $i \in V$ **do**
    $\mathcal{N}_i \leftarrow$ the set of neighbors of i in G
    for each node j in $\mathcal{N}_i$ **do**
        $\Delta Q_j \leftarrow$ the change in modularity when i is assigned to the community of j
    end for
    $j^* \leftarrow$ the neighbor that gives the most positive change in modularity ΔQ_{j^*}
    if $\Delta Q_{j^*} > 0$ **then**
        Assign node i to the community of j^*
    else
        Keep node i in its current community
    end if
end for

Phase 2 (Community Aggregation)

Input: a graph $G = (V_G, E_G)$ with a vertex set V_G and an edge set E_G; and the communities from Phase 1.

Output: a graph $H = (V_H, E_H)$ where each node now represents a community in the Phase 1 graph G.

$V_H \leftarrow \emptyset$
$E_H \leftarrow \emptyset$
for each community C **do**
    $V_H \leftarrow V_H \cup \{C\}$
end for
for each community C **do**
    for each community C' **do**
        $e \leftarrow \{C, C'\}$
        if $e \notin E_H$ **then**
            $E_H \leftarrow E_H \cup \{e\}$ (including a self-edge if $C = C'$)
            $w_e \leftarrow \sum_{\substack{v \in C \\ u \in C'}} w_{v,u}$. ▷ w_e is the weight assigned to the edge e in the graph H. Initially, $w_{v,u} = 1$ for all v and u.
        end if
    end for
end for

Again, we repeat phases 1 and 2 until no change in modularity occurs. We call each iteration of phases 1 and 2 one pass of the algorithm. Figure 1 below illustrates the Louvain algorithm.

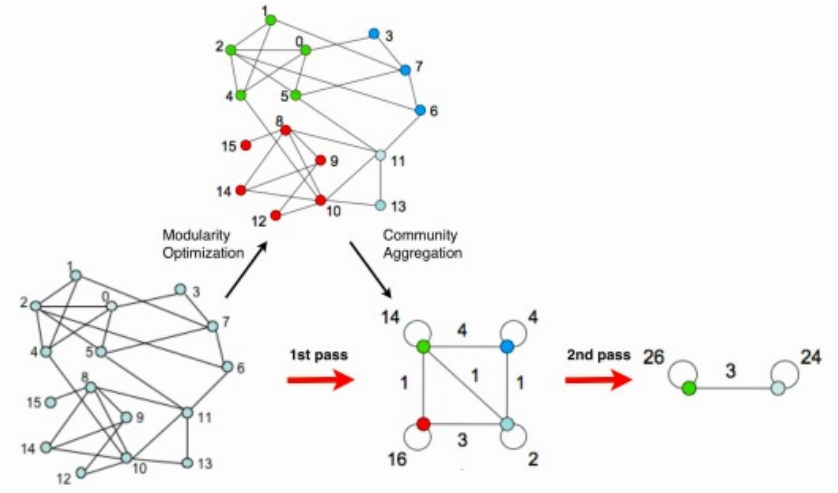

Figure 1: Example from Blondel et al. showing the two phases of the Louvain algorithm

(a) **[9 points]** Consider a node i that is in a community all by itself. Let C represent an existing community in the graph. Node i feels lonely and decides to move into the community C. This situation can be modeled by a graph (Figure 2) with C represented by a single node. The sum of the weights of the edges between i and a vertex in C is $k_{i,in}/2$, i.e. the community of i and C are connected by an edge of weight $k_{i,in}/2$. The sum of weights of edges incident to i (i.e. its weighted degree) is k_i, the sum of weights of edges incident to a vertex in C is Σ_{tot}, and $\Sigma_{in} = \sum_{j,k \in C} w_{j,k}$ is the sum of the weights of the edges with both endpoints in C. As always, $2m = \sum A_{ij}$ is the sum of all entries in the adjacency matrix. To begin with, C and i are in separate communities (colored green and red respectively). The third node represents the remainder of the graph.

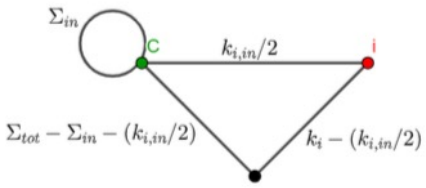

Figure 2: Before merging, i is an isolated node and C is a community. The rest of the graph is represented by a single node.

Prove that the modularity gain seen when i merges with C (i.e., the change in modularity after they merge into one community) is given by:

$$\Delta Q = \left[\frac{\Sigma_{in} + k_{i,in}}{2m} - \left(\frac{\Sigma_{tot} + k_i}{2m} \right)^2 \right] - \left[\frac{\Sigma_{in}}{2m} - \left(\frac{\Sigma_{tot}}{2m} \right)^2 - \left(\frac{k_i}{2m} \right)^2 \right].$$

Note that this expression gives us a computationally efficient way to compute the modularity changes in phase 1.

Hint: apply the community aggregation step of the Louvain algorithm to simplify the calculations.

(b) **[12 points]** Consider the graph G in Figure 3, with 4 cliques of 4 nodes each, arranged in a ring. Assume all the edges have the same weight value 1. There exists exactly one edge between any two adjacent cliques. We will manually (by hand) inspect the results of the Louvain algorithm on this network.

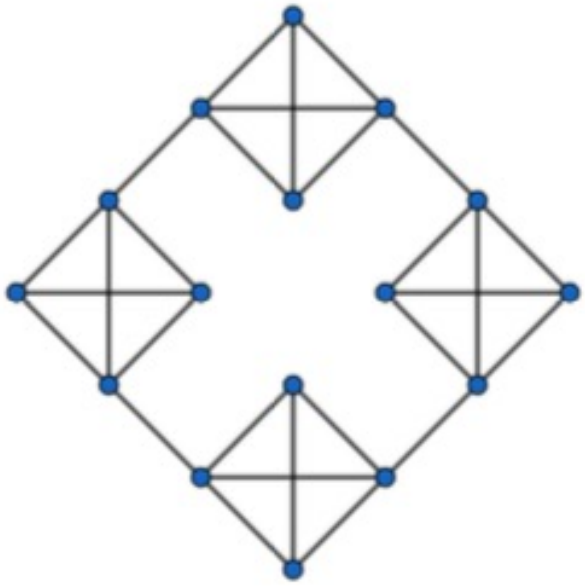

Figure 3: G is a subgraph. The whole graph has 16 nodes (4 cliques with 4 nodes per clique)

1. **[4 points]** The first phase of modularity optimization detects each clique as a single community, so there are 4 communities in total. Thus, the graph H output by the first pass of the Louvain algorithm is a graph with four nodes, each corresponding to one of the four cliques in G. What are the weights of each edge in the graph H? Explain.
   Hint: note that the symmetry of the ring structure simplifies the calculation.
2. **[3 points]** Derive the modularity of the graph H after the first pass of the Louvain algorithm.
3. **[5 points]** Show mathematically that the modularity of H does not change in the second pass of the algorithm, hence the algorithm terminates.
   Hint: due to the symmetry in H, you only need to calculate a single value of ΔQ. You may either calculate the modularity directly or extend the result of part (a).

(c) **[14 points]** Modularity optimization often fails to identify communities smaller than a certain scale, which is known as the **resolution limit problem**. We illustrate this

problem using a dataset with ground-truth communities; that is, we have labels for the "true" communities of the graph. We provide the following undirected YouTube social network. In the YouTube social network, users can form friendships with each other and users can create groups which other users can join. We consider such user-defined groups as ground-truth communities.

We are interested in quantifying how "good" the communities chosen by modularity by evaluating them via a goodness metric, which we present below. Here we compare 2 scoring functions: (1) *modularity* (higher is better) (2) *cut ratio* (lower is better): $f(S) = \frac{c_S}{n_S(n-n_S)}$, where c_s is the number of edges crossing the boundary of community S, n_S is the number of nodes in S and n is the number of nodes in the entire graph. Our goodness metric is *density* $g(S) = \frac{2m_S}{n_S(n_S-1)}$, where m_S is the number of edges within S.

Note that density favors small, highly connected communities; hence, we expect that scoring functions that do poorly with smaller communities will not perform well with respect to this metric, and that scoring functions that can accurately identify smaller communities will have strong performance with respect to this goodness metric. Thus, this creates a good test case for the resolution limit of *modularity*. From the definition, we could see cut ratio has already taken community size into account.

We run the following experiment: the file `youtube_community_top1000.txt` in the folder `louvain/data` contains the top 1000 ground-truth communities. For each community scoring function f, we rank the ground-truth communities by decreasing score of f. So, lower values of rank correspond to the "better" communities by each scoring function, whereas higher values of rank correspond to the "worse" communities under each scoring function. We measure the cumulative running average value of the goodness metric g of the top-k ground-truth communities under the ordering induced by f. Intuitively, a perfect community scoring function would rank the communities in decreasing order of the goodness metric, and thus the cumulative running average of the goodness metric would decrease monotonically with k.

You have been provided a skeleton in the file `PartitionQuality.py` in the folder `louvain/code`. Your task is to complete the functions `community_modularity`, `cut_ratio`, and `density`. Then, run the file, which executes the functions you have written and applies them to the YouTube dataset. Submit the plot `density.jpg` produced by the code as part of your write-up. Interpret the plot you produce. Which metrics are performing better for this dataset?
Hint: for the first ground-truth community in `youtube_community_top1000.txt`, the modularity is approximately 2.15×10^{-5}, the cut ratio is approximately 1.39×10^{-4}, and the density is approximately 3.62×10^{-2}.

What to submit

(i) Proof of 2(a).

(ii) Answers to the 3 subparts of 2(b).

(iii) The plot `density.jpg` produced by your code and an interpretation thereof. [2(c)]

(iv) Upload your completed implementation of `PartitionQuality.py` to Gradescope.

3 Spectral Clustering (40 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, $\overline{S} = V \backslash S$:

$$\text{cut}(S) = \sum_{i \in S, j \in \overline{S}} A_{ij}. \tag{1}$$

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:

$$\text{vol}(S) = \sum_{i \in S} d_i, \tag{2}$$

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

$$\text{NCUT}(S) = \frac{\text{cut}(S)}{\text{vol}(S)} + \frac{\text{cut}(\overline{S})}{\text{vol}(\overline{S})} \tag{3}$$

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 [20 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:

$$x_S^{(i)} = \begin{cases} \sqrt{\frac{\text{vol}(\overline{S})}{\text{vol}(S)}} & \text{if } i \in S \\ -\sqrt{\frac{\text{vol}(S)}{\text{vol}(\overline{S})}} & otherwise \end{cases} \tag{4}$$

To clarify (because the font may not be clear), in Equation 4, 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 [11 points]

Based on the facts we just proved about x chosen as per Equation 4, 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 4. 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 4 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:

$$\begin{array}{ll} \underset{x \in \mathbb{R}^n}{\text{minimize}} & \frac{x^\top L x}{x^\top D x}, \\ \text{subject to} & x^\top D e = 0, \\ & x^\top D x = 2m. \end{array} \tag{5}$$

Show that a minimizer of the optimization problem 5 is

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

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

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 [9 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 G into two clusters, and let $y \in \{1, -1\}^n$ be an assignment vector for a set S:

$$y_i = \begin{cases} 1 & \text{if } i \in S \\ -1 & \text{otherwise} \end{cases} \tag{6}$$

Then, the *modularity* of the assignment y is

$$Q(y) = \frac{1}{2m} \sum_{i,j=1}^{n} \left[A_{ij} - \frac{d_i d_j}{2m} \right] \delta(y_i, y_j). \tag{7}$$

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 6. Prove that

$$Q(y) = \frac{1}{2m} \left(-2 \cdot \text{cut}(S) + \frac{1}{m} \text{vol}(S) \cdot \text{vol}(\overline{S}) \right) \tag{8}$$

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 5 is $x^* = D^{-1/2}v$ [4(b)]

iii. Proof of Equation 8 [4(c)]