CS224W: Machine Learning with Graphs **Fall 2019**

Homework 1

Due 11:59pm PDT Thursday October 10, 2019

This problem set should be completed individually.

General Instructions

These questions require thought, but do not require long answers. Please be as concise as possible. You are allowed to take a maximum of 1 late period (see the information sheet at the end of this document for the definition of a late period).

Submission instructions: You should submit your answers and code via Gradescope. There will be a separate submission assignment for written and code.

Submitting answers: Prepare answers to your homework in a single PDF file and submit it via Gradescope to the HW1 (Written) assignment. Make sure that the answer to each sub-question is on a *separate, single page*. The number of the question should be at the top of each page. Please use the submission template files included in the bundle to prepare your submission. Failure to use the submission template file will result in a reduction of 2 points from your homework score.

Information sheet: Fill out the information sheet located at the end of the submission template file, and sign it in order to acknowledge the Honor Code (if typesetting the homework, you may type your name instead of signing). This should be the last page of your submission. Failure to fill out the information sheet will result in a reduction of 2 points from your homework score.

Submitting code: Upload your code on Gradescope to the HW1 (Code) assignment. Put the code for each question in a separate single Python file named `q[question number].py`. (For example, all code for all parts of question 1 should go in file `q1.py`.) Then upload a zip file containing all the Python files via Gradescope to the HW1 (Code) assignment. Failure to submit your code will result in reduction of all points for that question from your homework score.

Homework survey: After submitting your homework, please fill out the Homework 1 Feedback Form. Respondents will be awarded extra credit.

Questions

1 Network Characteristics [25 points]

One of the goals of network analysis is to find mathematical models that characterize real-world networks and that can then be used to generate new networks with similar properties. In this problem, we will explore two famous models—Erdős-Rényi and Small World—and compare them to real-world data from an academic collaboration network. Note that in this problem all networks are *undirected*. You may use the starter code in hw1-q1-starter.py for this problem.

- *Erdős-Rényi Random graph ($G(n, m)$ random network):* Generate a random instance of this model by using $n = 5242$ nodes and picking $m = 14484$ edges at random. Write code to construct instances of this model, i.e., do not call a SNAP function.
- *Small-World Random Network:* Generate an instance from this model as follows: begin with $n = 5242$ nodes arranged as a ring, i.e., imagine the nodes form a circle and each node is connected to its two direct neighbors (e.g., node 399 is connected to nodes 398 and 400), giving us 5242 edges. Next, connect each node to the neighbors of its neighbors (e.g., node 399 is also connected to nodes 397 and 401). This gives us another 5242 edges. Finally, randomly select 4000 pairs of nodes not yet connected and add an edge between them. In total, this will make $m = 5242 \cdot 2 + 4000 = 14484$ edges. Write code to construct instances of this model, i.e., do not call a SNAP function.
- *Real-World Collaboration Network:* Download this undirected network from `http://snap.stanford.edu/data/ca-GrQc.txt.gz`. Nodes in this network represent authors of research papers on the arXiv in the General Relativity and Quantum Cosmology section. There is an edge between two authors if they have co-authored at least one paper together. Note that some edges may appear twice in the data, once for each direction. Ignoring repeats and self-edges, there are 5242 nodes and 14484 edges. (Note: Repeats are automatically ignored when loading an (un)directed graph with SNAP's `LoadEdgeList` function).

1.1 Degree Distribution [12 points]

Generate a random graph from both the Erdős-Rényi (i.e., $G(n, m)$) and Small-World models and read in the collaboration network. Delete all of the self-edges in the collaboration network (there should be 14,484 total edges remaining).

Plot the degree distribution of all three networks *in the same plot* on a log-log scale. In other words, generate a plot with the horizontal axis representing node degrees and the vertical axis representing the proportion of nodes with a given degree (by "log-log scale" we mean that both the horizontal and vertical axis must be in logarithmic scale). In one to two sentences, describe one key difference between the degree distribution of the collaboration network and the degree distributions of the random graph models.

1.2 Clustering Coefficient [13 points]

Recall that the local clustering coefficient for a node v_i was defined in class as

$$C_i = \begin{cases} \frac{2|e_i|}{k_i \cdot (k_i - 1)} & k_i \geq 2 \\ 0 & \text{otherwise,} \end{cases}$$

where k_i is the degree of node v_i and e_i is the number of edges between the neighbors of v_i. The *average clustering coefficient* is defined as

$$C = \frac{1}{|V|} \sum_{i \in V} C_i.$$

Compute and report the average clustering coefficient of the three networks. For this question, write your own implementation to compute the clustering coefficient, instead of using a built-in SNAP function.

Which network has the largest clustering coefficient? In one to two sentences, explain. Think about the underlying process that generated the network.

What to submit

Page 1:
- Log-log degree distribution plot for all three networks (in same plot)
- One to two sentences description of a difference between the collaboration network's degree distribution and the degree distributions from the random graph models.

Page 2:
- Average clustering coefficient for each network.
- Network that has the largest average clustering coefficient.
- One to two sentences explaining why this network has the largest average clustering coefficient.

2 Structural Roles: Rolx and ReFex [25 points]

In this problem, we will explore the structural role extraction algorithm Rolx and its recursive feature extraction method ReFex. As part of this exploration, we will work with a dataset representing a scientist co-authorship network, which can be dowloaded at http://www-personal.umich.edu/~mejn/netdata/netscience.zip. [1] Although the graph is weighted, for simplicity we treat it as **undirected and unweighted** in this problem.

Feature extraction consists of two steps; we first extract basic local features from every node, and we subsequently aggregate them along graph edges so that global features are also obtained. Collectively, feature extraction constructs a matrix $V \in \mathbb{R}^{n \times f}$ where for each of the n nodes we have f features to cover local and global information. Rolx extracts node roles from that matrix.

2.1 Basic Features [5 points]

We begin by loading the graph G provided in the bundle and computing three basic features for the nodes. For each node v, we choose 3 basic local features (in this order):

1. the degree of v, i.e., $\deg(v)$;
2. the number of edges in the egonet of v, where egonet of v is defined as the subgraph of G induced by v and its neighborhood;
3. the number of edges that connect the egonet of v and the rest of the graph, i.e., the number of edges that enter or leave the egonet of v.

We use $\tilde{V}_u$ to represent the vector of the basic features of node u. For any pair of nodes u and v, we can use cosine similarity to measure how similar two nodes are according to their feature vectors x and y:

$$\text{Sim}(x, y) = \frac{x \cdot y}{||x||_2 \cdot ||y||_2} = \frac{\sum_i x_i y_i}{\sqrt{\sum_i x_i^2} \cdot \sqrt{\sum_i y_i^2}};$$

Also, when $||x||_2 = 0$ or $||y||_2 = 0$, we defined $\text{Sim}(x, y) = 0$.

Compute the basic feature vector for the node with ID 9, and report the top 5 nodes that are most similar to node 9 (excluding node 9). As a sanity check, no element in $\tilde{V}_9$ is larger than 10.

2.2 Recursive Features [8 points]

In this next step, we recursively generate some more features. We use *mean* and *sum* as aggregation functions.

[1] We provide a binary file named *hw1-q2.graph* for you to directly load into snap by `G = snap.TUNGraph.Load(snap.TFIn("hw1-q2.graph"))` . You are welcome to either use this file or load from raw data yourself.

Initially, we have a feature vector $\tilde{V}_u \in \mathbb{R}^3$ for every node u. In the first iteration, we concatenate the mean of all u's neighbors' feature vectors to $\tilde{V}_u$, and do the same for *sum*, i.e.,

$$\tilde{V}_u^{(1)} = \left[\tilde{V}_u;\ \frac{1}{|N(u)|}\sum_{v\in N(u)}\tilde{V}_v;\ \sum_{v\in N(u)}\tilde{V}_v\right] \in \mathbb{R}^9,$$

where $N(u)$ is the set of u's neighbors in the graph. If $N(u) = \emptyset$, set the mean and sum to 0.

After K iterations, we obtain the overall feature matrix $V = \tilde{V}^{(K)} \in \mathbb{R}^{3^{K+1}}$.

For this exercise, run $K = 2$ iterations, and report the top 5 nodes that are most similar to node 9 (excluding node 9). If there are ties, e.g. 4th, 5th, and 6th have the same similarity, report any of them to fill up the top-5 ranking. As a sanity check, the similarities between the reported nodes and node 9 are all greater than 0.9. [5 points]

Compare your obtained top 5 nodes with previous results from 2.1. In particular, are there common nodes? Are there different nodes? In one sentence, why would this change? [3 points]

2.3 Role Discovery [12 points]

In this part, we explore more about the graph according to the recursive feature vectors of nodes and node similarity.

(a) Produce a 20-bin histogram to show the distribution of cosine similarity between node 9 and any other node in the graph (according to their recursive feature vectors). Note here that the x-axis is cosine similarity with node 9, and the y-axis is the number of nodes. [3 points]

According to the histogram, can you spot some *groups / roles*? How many can you spot? (*Clue: look for the spikes!*) [2 points]

(b) For these groups / roles in the cosine similarity histogram, take one node u from each group to examine the feature vector, and draw the subgraph of the node based on its feature vector. You can draw the subgraph by hand, or you can use libraries such as networkx or graphviz.

For these drawings, you should use the local features for u, and pay attention to the features aggregated from its 1-hop neighbors, but feel free to ignore further features if they are difficult to incorporate. Also, you should not draw nodes that are more than 3-hops away from u. [6 points]

Briefly argue how different structural roles are captured in 1-2 sentences. [1 point]

What to submit

Page 3:
- The basic feature vector for node with ID 9, and the nodes most similar to node 9 in terms of cosine similarity. [5 points]

Page 4:
- Top 5 nodes that are closest to node 9 with respect to cosine similarity. [5 points]
- Whether there are common and different nodes in the closest nodes reported for basic feature vectors and full feature vectors. Also include a one-sentence explanation about why this might change from the results in 2.1. [3 points]

Page 5:
- (a) A histogram for similarity distribution. Whether you can spot some groups, and if yes, how many. [5 points]

Page 6:
- (b) Several hypothetical subgraphs, one for each group in the cosine similarity histogram. Also include a brief argument about how different structural roles are captured. [7 points]

3 Community detection using the Louvain algorithm [25 points]

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

Communities are a fundamental aspect of several networks. However, it is often not an easy task to come up with an "optimal" grouping of nodes into communities. Through this problem, we will explore some properties of the Louvain algorithm for community detection (so named because the authors were all affiliated with the University of Louvain in Belgium at some point). The original paper from Blondel et al. is available here: https://arxiv.org/pdf/0803.0476.pdf. If you are stuck on this question at any point please refer to the paper; there is a good chance that you will find what you seek there.

We will first explore the idea of modularity. The modularity of a weighted graph is a measure that compares the density of edges within a community to the density of edges between communities. Formally, we define the modularity Q for a given graph as follows:

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

Here $2m = \Sigma A_{ij}$ is the sum of all entries in the adjacency matrix, A_{ij} represents the $(i,j)^{th}$ entry of the adjacency matrix, d_i represents the degree of node i, $\delta(c_i, c_j)$ is 1 when i and j are in the same community ($c_i = c_j$) and 0 otherwise. Note that we treat communities as disjoint. In other words, a given node from a graph can only belong to one community in that graph.

The modularity of a graph lies in the range $[-1, 1]$. Maximizing the modularity of a given graph is a computationally hard problem, so we try different heuristics for this purpose. One such heuristic is the **Louvain algorithm**. This algorithm outperforms many similar algorithms in terms of both speed as well as maximum modularity obtained. We will run a few steps of the algorithm on a couple of example networks to gain some insights about its behavior and properties.

Each pass of the algorithm has two phases, and proceeds as follows:

- **Phase 1 (Modularity Optimization):** Start with each node in its own community.
- For each node i, go over all the neighbors j of i. Calculate the change in modularity when i is moved from its present community to j's community. Find the neighbor j_m for which this process causes the greatest increase in modularity, and assign i to j_m's community (break ties arbitrarily). If there is no positive increase in modularity during this process, keep i in its original community.
- Repeat the above process for each node (going over nodes multiple times if required) until there is no further maximization possible (that is, each node remains in its community). This is the end of Phase 1.
- **Phase 2 (Community Aggregation):** Once Phase 1 is done, we contract the original graph G to a new graph H. Each community found in G after Phase 1 becomes a node in H. The weights of edges in between 2 nodes in H are given by the sums of the weights between the respective 2 communities in G. The weights of edges within a community in G sum up to form a self-edge of the same weight in H (be careful while calculating self-edge weights, note that you will have to go once over each original node within the community in G). This is the end of Phase 2. Phase 1 and Phase 2 together make up a single pass of the algorithm.

- Repeat Phase 1 and Phase 2 again on H and keep proceeding this way until no further improvement is seen (you will reach a point where each node in the graph stays in its original community to maximize modularity). The final modularity value is a heuristic for the maximum modularity of the graph.

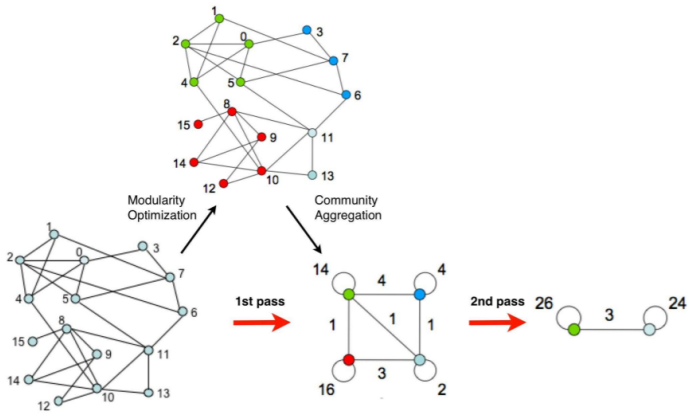

Figure 1: Example from Blondel et al. showing the two phases of the Louvain algorithm. Note how weights for self-edges are assigned in the Community Aggregation phase.

An example taken from Blondel et al. (Figure 1) illustrates the two phases of the algorithm. Note how weights for self-edges are assigned in the Community Aggregation phase - we will need to use this later. The weight of the self-edge formed by merging all nodes in a community K would be given by $\Sigma_{i\in K}\Sigma_{j\in K}A_{ij}$ - you can verify this for yourself by checking in Figure 1 that node 11 and 13 have an edge of weight 1 between them, but the corresponding self-edge has a weight of 2.

3.1 Modularity gain when an isolated node moves into a community [4 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, we will inspect the change in modularity when this happens.

This situation can be modeled by a graph (Figure 2) with C being represented by a single node. C has a self-edge of weight Σ_{in}. There is an edge between i and C of weight $k_{i,in}/2$ (to stay consistent with the notation of the paper). The total degree of C is Σ_{tot} and the degree of i is k_i. As always, $2m = \Sigma A_{ij}$ 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). 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:

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

Hint: Using the community aggregation step of the Louvain method may make computation easier.

In practice, this result is used while running the Louvain algorithm (along with a similar related result) to make incremental modularity computations much faster.

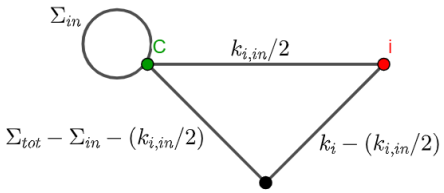

Figure 2: Before merging, i is an isolated node and C represents an existing community. The rest of the graph can be treated as a single node for this problem.

3.2 Louvain algorithm on a 16 node network [10 points]

Consider the graph G (Figure 3), with 4 cliques of 4 nodes each arranged in a ring. Assume all the edges have same weight value 1. There exists exactly one edge between any two adjacent cliques. We will manually inspect the results of the Louvain algorithm on this network. The first phase of modularity optimization detects each clique as a single community (giving 4 communities in all). After the community aggregation phase, the new network H will have 4 nodes.

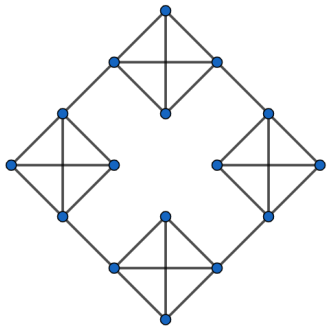

Figure 3: G is a graph with 16 nodes (4 cliques with 4 nodes per clique).

- What is the weight of any edge between two distinct nodes in H? [1 point]
- What is the weight of any self-edge in H? [2 point]
- What is the modularity of H (with each node in its own community)? The easiest way to calculate modularity would be to directly apply the definition from 1 to H (also holds for upcoming questions of this type). [2 points]

Spoiler alert: In this network, this is the maximum modularity and the algorithm will terminate here. However, assume that we wanted to contract the graph further into a two node network (call it J) by grouping two adjacent nodes in H into one community and the other two adjacent nodes into another community, and then aggregating (following the same rules of the community aggregation phase).

- What is the weight of any edge between two distinct nodes in J? [1 point]

- What is the weight of any self-edge in J? [2 point]
- What is the modularity of J (with each node in its own community)? [2 points]

As expected, the modularity of J is less than the modularity of H.

3.3 Louvain algorithm on a 128 node network [10 points]

Now consider a larger version of the same network, with 32 cliques of 4 nodes each (arranged in a ring as earlier); call this network G_{big}. Again, assume all the edges have same weight value 1, and there exists exactly one edge between any two adjacent cliques. The first phase of modularity optimization, as expected, detects each clique as a single community. After aggregation, this forms a new network H_{big} with 32 nodes.

- What is the weight of any edge between two distinct nodes in H_{big}? [1 point]
- What is the weight of any self-edge in H_{big}? [2 point]
- What is the modularity of H_{big} (with each node in its own community)? [2 points]

After what we saw in the earlier example, we would expect the algorithm to terminate here. However (spoiler alert again), that doesn't happen and the algorithm proceeds. The next phase of modularity optimization groups H_{big} into 16 communities with two adjacent nodes from H_{big} in each community. Call the resultant graph (after community aggregation) J_{big}.

- What is the weight of any edge between two distinct nodes in J_{big}? [1 point]
- What is the weight of any self-edge in J_{big}? [2 point]
- What is the modularity of J_{big} (with each node in its own community)? [2 points]

This particular grouping of communities corresponds to the maximum modularity in this network (and not the one with one clique in each community). The community grouping that maximizes the modularity here corresponds to one that would not be considered intuitive based on the structure of the graph.

3.4 What just happened? [1 point]

Explain (in a few lines) why you think the algorithm behaved the way it did for the larger network (you don't need to prove anything rigorously, a rough argument will do). In other words, what might have caused modularity to be maximized with an "unintuitive" community grouping for the larger network?

What to submit:

Page 7:
- Proof for change in modularity when node i moves to community C.

Page 8:
- Weight of edge between two distinct nodes in H.

- Weight of self-edge in H.
- Modularity of H with each node in its own community.
- Weight of edge between two distinct nodes in J.
- Weight of self-edge in J.
- Modularity of J with each node in its own community.

Page 9:
- The same as above, except for H_{big} and J_{big}.

Page 10:
- Explanation of algorithm behavior in the larger network (a few lines).

4 Spectral clustering [25 points]

This question derives a spectral clustering algorithm that we then use to analyze a real-world dataset. These algorithms use eigenvectors of matrices associated with the graph. You may find this handout https://bit.ly/2l0dXCL on graph clustering to be useful for additional background information.

Let's first discuss some notation:

- Let $G = (V, E)$ be a simple (that is, no self- or multi- edges) undirected, connected graph with $n = |V|$ and $m = |E|$.
- A is the adjacency matrix of the graph G, i.e., A_{ij} is equal to 1 if $(i, j) \in E$ and equal to 0 otherwise.
- D is the diagonal matrix of degrees: $D_{ii} = \sum_j A_{ij}$ = the degree of node i.
- We define the *graph Laplacian* of G by $L = D - A$.

For a set of nodes $S \subset V$, we will measure the quality of S as a cluster with a "cut" value and a "volume" value. We define the cut of the set S to be the number of edges that have one end point in S and one end point in the complement set $\bar{S} = V \backslash S$:

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

Note that the cut is symmetric in the sense that $\text{cut}(S) = \text{cut}(\bar{S})$. The *volume* of S is simply the sum of degrees of nodes in S:

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

where d_i is the degree of node i.

4.1 A Spectral Algorithm for Normalized Cut Minimization: Foundations [10 points]

We will try to find a set S with a small normalized cut value:

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

Intuitively, a set S with a small normalized cut value must have few edges connecting to the rest of the graph (making the numerators small) as well as some balance in the size of the clusters (making the denominators large).

Define the assignment vector x for some set of nodes S such that

$$x_i = \begin{cases} \sqrt{\dfrac{\text{vol}(\bar{S})}{\text{vol}(S)}} & i \in S \\ -\sqrt{\dfrac{\text{vol}(S)}{\text{vol}(\bar{S})}} & i \in \bar{S} \end{cases} \tag{4}$$

Prove the properties below.

Note: There are many ways to prove the following properties, and we provide some hints for one of the ways. You do not necessarily need to use the provided hints for your proof.

(i) $L = \sum_{(i,j)\in E}(e_i - e_j)(e_i - e_j)^T$, where e_k is an n-dimensional column vector with a 1 at position k and 0's elsewhere. Note that we aren't summing over the entire adjacency matrix and only count each edge once.

(ii) $x^T Lx = \sum_{(i,j)\in E}(x_i - x_j)^2$. *Hint: Apply the result from part (i).*

(iii) $x^T Lx = c \cdot \text{NCUT}(S)$ for some constant c (in terms of the problem parameters). *Hint: Rewrite the sum in terms of S and $\bar{S}$.*

(iv) $x^T De = 0$, where e is the vector of all ones.

(v) $x^T Dx = 2m$.

4.2 Normalized Cut Minimization: Solving for the Minimizer [8 points]

Since $x^T Dx$ is just a constant ($2m$), we can formulate the normalized cut minimization problem in the following way:

$$\begin{aligned} &\underset{S\subset V,\, x\in\mathbb{R}^n}{\text{minimize}} && \frac{x^T Lx}{x^T Dx} \\ &\text{subject to} && x^T De = 0,\ x^T Dx = 2m,\ x \text{ as in Equation 4} \end{aligned} \tag{5}$$

The constraint that x takes the form of Equation 4 makes the optimization problem NP-hard. We will instead use the "relax and round" technique where we relax the problem to make the optimization problem tractable and then round the relaxed solution back to a feasible point for the original problem. Our relaxed problem will eliminate the constraint that x take the form of Equation 4 which leads to the following relaxed problem:

$$\begin{aligned} &\underset{x\in\mathbb{R}^n}{\text{minimize}} && \frac{x^T Lx}{x^T Dx} \\ &\text{subject to} && x^T De = 0,\ x^T Dx = 2m \end{aligned} \tag{6}$$

Show that the minimizer of Equation 6 is $D^{-1/2}$v, where v is the eigenvector corresponding to the second smallest eigenvalue of the *normalized graph Laplacian* $\tilde{L} = D^{-1/2}LD^{-1/2}$. Finally, to round the solution back to a feasible point in the original problem, we can take the indices of all positive entries of the eigenvector to be the set S and the indices of all negative entries to be $\bar{S}$.

Hint 1: Make the substitution $z = D^{1/2}x$.

Hint 2: Note that e is the eigenvector corresponding to the smallest eigenvalue of L.

Hint 3: The normalized graph Laplacian $\tilde{L}$ is symmetric, so its eigenvectors are orthonormal and form a basis for $\mathbb{R}^n$. This means we can write any vector x as a linear combination of orthonormal eigenvectors of $\tilde{L}$.

4.3 Relating Modularity to Cuts and Volumes [7 points]

In Problem 3, 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's consider a partitioning of our graph A into 2 clusters, and let $y \in \{1, -1\}^n$ be an assignment vector for a set S:

$$y_i = \begin{cases} 1 & \text{if } i \in S \\ -1 & \text{if } i \in \bar{S} \end{cases} \tag{7}$$

Then, the *modularity* of the assignment y is

$$Q(y) = \frac{1}{2m} \sum_{1 \leq i,j \leq n} \left[A_{ij} - \frac{d_i d_j}{2m} \right] I_{y_i = y_j}. \tag{8}$$

Let y be the assignment vector in Equation 7. Prove that

$$Q(y) = \frac{1}{2m} \left(-2 \cdot \text{cut}(S) + \frac{1}{m} \text{vol}(S) \cdot \text{vol}(\bar{S}) \right) \tag{9}$$

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

Page 11:
- Proofs of the five properties.

Page 12:
- Proof of the minimizer.

Page 13:
- Proof of the relationship between modularity and cuts and volumes.