ECEN5322: Search Engines and Analysis of High Dimensional Datasets 10/30/2015

Lab Report 3

Community Detection in Networks *Author: Prasanth Prahladan(100817764)*

1 Experiments

1.1 Planted Partition Model

Q 13. Matlab function that takes p,q,n as input and generates the adjacency matrix of the planted partition model.

```matlab
function [A, partitionIndicatorVec] = getPartitionGraphModel(n,p,q)
%{
Q 13.
n : total #nodes in G
p : probability of link between two vertices inside Cluster
q : probability of link edges between two vertices in opposite clusters
%}

 if (mod(n,2) == 0 ) % n is EVEN
%     generate Permutation Matrix, T
     I = eye(n);
     ix = randperm (n);
     T = I(ix,:);

%     generate adjaceny matrix A of a planted partition over n nodes
     n2 = n/2;
     P = random('bino', 1, p, n2, n2); % upper left block
     dP2 = random('bino', 1, p, n2, 1); % diagonal of the lower right block
     Q = random('bino', 1, q, n2, n2); % upper right block
%     carve the two triangular and diagonal matrices that we need
     U = triu(P, 1);
     L = tril(P,-1);
     dP = diag(P);
     B0 = U + U' + diag(dP);
     B1 = Q;
     B2 = Q';
     B3 = L + L' + diag(dP2);
     B =[B0 B1;B2 B3];

     comm1 = ones(1,n2);
     originalCluster = [comm1 -1*comm1];


%  PERMUTE THE NODES
%     Re-index Nodes of the graph.
%     B*T' -> exchg columns; T*M -> exchg rows
     A = T*B*T';

%   Obtain the True_Cluster_NodeID for Graph A: Permute them
     partitionIndicatorVec  = originalCluster(ix);
%%--------------------------------------------------
 else
    warning('n == ODD!') ;
end
end
```

Q 14. Implementation of Partition Algorithm

```matlab
function partitionIndicatorVec = runPartitionAlgo(A)
%{
```

```
Algorithm Partition
* compute the second dominant eigenvector , v2, of A, associated with the second largest
eigenvalue ?2.
* for i = 1 to n
if the coordinate i of v2 is positive ,(v2)_i > 0, then %what if its' ZERO?
assign node wi to community 1
else
assign node i to community 2.
end
end

partitionIndicatorVec := {1(partition A), −1(partition B)}

%}


[V, D] = eigs(A,2);

vec = V(:,2)';
pos = (vec >0);
neg = (vec <0);
partitionIndicatorVec = pos − neg;

end
```

Q 15. Computing Overlap between the True-Partition and Predicted/Estimated-Partitions

$$\omega_i = \begin{cases} 1 & \text{if i belongs to partition 1} \\ -1 & \text{if i belongs to partition 2} \end{cases} \tag{1}$$

$$\tilde{\omega_i} = \begin{cases} 1 & \text{if } (v_2)_i > 0, \\ -1 & \text{otherwise} \end{cases} \tag{2}$$

$$rawoverlap = max\left(\sum_{i=1}^{n} \delta_{\omega_i,\tilde{\omega_i}}, \sum_{i=1}^{n} \delta_{-\omega_i,\tilde{\omega_i}}\right) \tag{3}$$

$$overlap = \frac{2}{n} rawoverlap - 1 \tag{4}$$

(a) Compute overlap score when $\tilde{\omega} = \omega$.

From (3) we obtain $rawoverlap = n$ when $\tilde{\omega} = \omega$. Substituting into (4), we obtain

$$overlap = \frac{2}{n}(n) - 1 = 1$$

(b) Prove that a random guess for the detection of the communities returns overlap 0.
A random guess for the detection of communities is a Binary vector of $\{-1, 1\}^n$ with each component chosen with equal probability (0.5). From the definition (3), we obtain $rawoverlap = \frac{n}{2}$. Thus, we obtain

$$overlap = \frac{2}{n}\frac{n}{2} - 1 = 0.$$

```
function  overlap = getPartitionOverlap(w1, w2)
%{
Q15.
Derive the Overlap Metric based on the
w1 : true partition vector
w2 : estimated partition vector
%}
n = length(w1);
del_w1_w2 = sum(w1 == w2);
```

```
del_minus_w1_w2 = sum(-w1 == w2);

rawoverlap = max(del_w1_w2, del_minus_w1_w2);

% random choice of w2 generates a non-zero overlap. Accounting for this:
overlap = (2/n)*rawoverlap - 1;          % Interpret: Prob. of successfully detecting
    communities.
end
```

Q 16/17. Dense and Sparse Communities

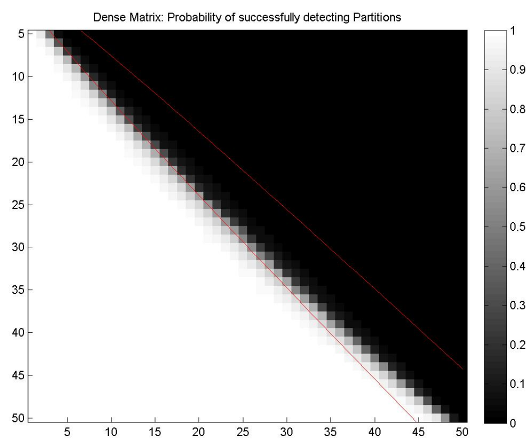

Figure 1: Dense Network: Probability of successfully detecting the partitions using the Partition-Algorithm. The x-axis corresponds to β and the y-axis corresponds to α. The decision boundaries are overlaid in red. The community-recovery algorithm is implemented only for case where $0 \leq q < p \leq 1 <$. The decision boundary lying inside the Dark region has (α, β) values that violate the $q < p$ requirement, and hence, can be ignored.

```
n = 300;
numTrials = 20;
alphaMax = 70;
betaMax = 50;
alphaMin = 5;
betaMin = 1;

Alpha = alphaMin:1:alphaMax;
Beta = betaMin:1:betaMax;

overlapMatrix = zeros(length(Alpha), length(Beta));

for i=1:length(Alpha)
    for j= 1:length(Beta)

        alpha = Alpha(i);
        beta = Beta(j);

%%          % Dense Matrix
%%          p = alpha/n*log(n);
%%          q = beta/n*log(n);

        % Sparse Matrix
        p = alpha/n;
        q = beta/n;

        overlapScore = zeros(1,numTrials);
```

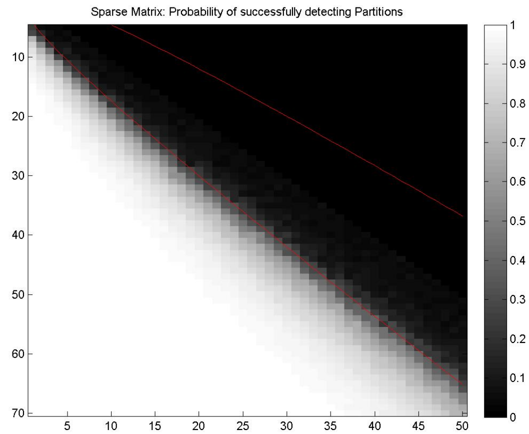

Figure 2: Sparse Network: Probability of successfully detecting the partitions using the Partition-Algorithm. The x-axis corresponds to b and the y-axis corresponds to a. The decision boundaries are overlaid in red. The community-recovery algorithm is implemented only for case where $0 \leq q < p \leq 1 <$. The decision boundary lying inside the Dark region has (a, b) values that violate the $q < p$ requirement, and hence, can be ignored.

```
            % ----------------------------------------------------------------
            % We need to run the algorithm only if q<p
            % We set the default values to zero!
            if (q<p)
             for iter = 1:numTrials
                 [A, w] = getPartitionGraphModel(n,p,q);
                 w_pred = runPartitionAlgo(A);
                 overlapScore(iter) = getPartitionOverlap(w, w_pred);
             end
            end
            % ----------------------------------------------------------------
             % store the avg. Overlap score for given (alpha,beta)
             overlapMatrix(i,j) = sum(overlapScore)/ numTrials;
        end
end
%%%
%%%% ----------------------------------------------------------
%%%% Plot the following curve on the
%%%% alpha - beta > sqrt(0.5*(alpha + beta))*2 / sqrt(log(n))
%%%% k = 2/sqrt(log n)
%%%% alpha > 0.5*( (2*beta + k^2/2)\pm k/2 sqrt(16*beta + k^2/2)     )
%%%
%%% k = 2/sqrt(log(n));
%%% Alpha1 = 0.5*((2*Beta + k^2/2) + (k/2)*sqrt(16*Beta + k^2));
%%% Alpha2 = 0.5*((2*Beta + k^2/2) - (k/2)*sqrt(16*Beta + k^2));
%%%% ----------------------------------------------------------

% ------------------------------------------------------------
% How does this change for the sparse matrix?
% SPARSE MATRIX BOUNDARIES
Alpha1 = (Beta + 1) + sqrt(1+ 4*Beta);
Alpha2 = (Beta + 1) - sqrt(1+ 4*Beta);

% ------------------------------------------------------------
close all
fig1 = figure(1)
% Plot the image as a GRAYSCALE
betaDim = [betaMin betaMax];
alphaDim = [alphaMin alphaMax];
```

```
img = imagesc(betaDim, alphaDim, overlapMatrix);
colormap(gray);
colorbar;
hold on
ax2 = plot(Beta,Alpha1,'r');
% colormap(ax2, parula )
hold on
ax3 = plot(Beta,Alpha2,'r');
% colormap(ax3, parula )
hold off
title('Sparse Matrix: Probability of successfully detecting Partitions')
% title('Dense Matrix: Probability of successfully detecting Partitions')
```

Q 18. Zachary's Karate Club

Overlap Score = 1. The code used for implementing the same is described below.

```
load zachary.mat

% From the question/visual graph: Identify true_Partition
nodeIDs = 1:34;
idx_teamA = [25 26 28 32 24 29 30 27 10 34 9 21 33 31 19 23 15 16]
idx_teamB = setdiff(1:34,idx_teamA)

truePartition = ones(size(nodeIDs));
truePartition(idx_teamB) = -1*ones(size(idx_teamB));


% Implement the algorithm to detect the partitions.

M = (A ~= 0 );
adjMatrix = zeros(size(A));
adjMatrix(M) = 1;

estPartition = runPartitionAlgo(adjMatrix);
cluster1 = (estPartition <0);
cluster1_idx_est = cluster1.*nodeIDs;

cluster2 = (estPartition >0);
cluster2_idx_est = cluster2.*nodeIDs;

overlapScore = getPartitionOverlap(truePartition, estPartition)
```
