L3S Research Center
Large Scale Data Mining, SS 2016
Dr. Avishek Anand
Solution to Assignment 6, due:

Problem 1.
Suppose we have a coin, which may not be a fair coin, and we flip it some number of times, seeing h heads and t tails.

1. If the probability p of getting a head on any flip is p, what is the MLE for p, in terms of h and t?
2. Suppose we are told that there is a 90% probability that the coin is fair (i.e., $p = 0.5$), and a 10% chance that $p = 0.1$. For what values of h and t is it more likely that the coin is fair?

Solution:

1. P(Head) $= p$ and P(Tail) $= 1 - p$. Let the number of flips be $n = h + t$. Note, we assume that each flip is an independent event. Probability of h heads and t tails in n flips is given by
$$P = nC_h\, p^h (1-p)^t \tag{1}$$
To estimate the parameter p, we use MLE.
$$\frac{\partial P}{\partial p} = 0 \tag{2}$$
$$(1-p)^t h p^{h-1} + (-1)t(1-p)^{t-1}p^h = 0 \tag{3}$$
Solving for p we get
$$p = \frac{h}{h+t} \tag{4}$$

Problem 2.

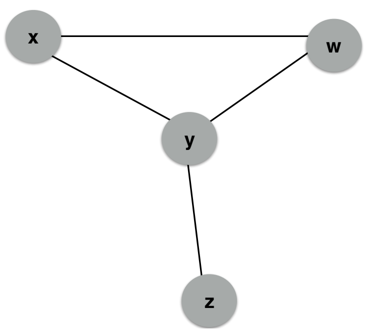

Figure 1: Community Graph

Compute the MLE for the graph in Figure 1 for the following guesses of the memberships of the two communities.

1. $C = \{w, x\}$; $C = \{y, z\}$.
2. $C = \{w, x, y, z\}$; $C = \{x, y, z\}$.

Solution:

1. $C_1 = w, x$ Let the probability of an edge be p_1 ;$C_2 = y, z$ Let the probability of an edge be p_2

   We use AGM to model the graph which implies that for a given pair of vertices u and v,

   $$P(u, v) = 1 - \prod_{c \in M_u \cap M_v} (1 - p_c) \tag{5}$$

   where the product is computed only if u and v are in the same community.

   For instance $P(w, x) = 1 - (1 - p_1) = p_1$. Note that $P(x, z) = P(w, z) = \epsilon$ since no edge exists between the vertices chosen in graph G.

   Now according to AGM we get:

   $$P(G|p_1, p_2) = P(x, w)P(x, y)P(w, y)P(y, z)(1 - P(x, z))(1 - P(x, z)) \tag{6}$$

   $$P(G|p_1, p_2) = p_1.1.1.p_2(1 - \epsilon)^2 \tag{7}$$

   $(1 - \epsilon)$ is close to 1 and can be ignored.

   From the above equation is clear that the probability of generating G from p_1 and p_2 is maximized when $p_1 = p_2 = 1$

2. $C_1 = w, x, y, z$ Let the probability of an edge be p_1; $C_2 = x, y, z$ Let the probability of an edge be p_2

   Note that $P(y, z) = P(x, y) = P(x, z) = 1 - ((1 - p_1)(1 - p_2))$

   $$P(G|p_1, p_2) = P(x, w)P(x, y)P(w, y)P(y, z)(1 - P(x, z))(1 - P(x, z)) \tag{8}$$

   $$P(G|p_1, p_2) = (p_1)^2(p_1 + p_2 - p_1p_2)^2(1 - (p_1 + p_2 - p_1p_2))(1 - p_1) \tag{9}$$

   You can use MLE to determine p_1 and p_2 since it is difficult to make conclusions from this equation directly. Note that this happens when you have nested communities when using AGM.

Problem 3. Suppose graphs are generated by picking a probability p and choosing each edge independently with probability p. For the graph of Figure 1 , what value of p gives the maximum likelihood of seeing that graph? What is the probability this graph is generated?
Solution:

$$P(G|p) = P(x, w)P(x, y)P(w, y)P(y, z)(1 - P(x, z))(1 - P(x, z)) \tag{10}$$

Probability of generating the graph is given by:

$$P(G|p) = p^4(1 - p)^2 \tag{11}$$

Log Likelihood of P is given by $4 \log p + 2 \log(1 - p)$

$$\frac{\partial L}{\partial p} = 0 \tag{12}$$

$$\frac{4}{p} + \frac{2}{1-p}(-1) = 0 \tag{13}$$

Solving for p we get $p = \frac{2}{3}$ Therefore, $P(G|p) = (\frac{2}{3})^4(\frac{1}{3})^2$

Problem 4. Compute the number of triangles and Clustering coefficient (for each node) of a

1. Complete graph (clique) with n vertices.
2. Complete bi-partite graph with left set with l and right set with m vertices.
3. Consider a node A in a graph G. A has exactly m neighbors with an edge probability between the neighbors being p. What is the expected value of the clustering coefficient for node A.

Solution:

1. A complete graph has an edge between every pair of vertices. Therefore the number of triangles is all possible combinations of 3 vertices which is nC_3
   Clustering Coefficient $cc(v) = \frac{\#\Delta' s\, on\, v}{d_v C_2}$

$$cc = \frac{n-1C_2}{n-1C_2} = 1 \tag{14}$$

2. The number of triangles in a bi-partite graph is 0. With our given definition of cc we get 0 again for the clustering co-efficient.

3.

$$cc(A) = \frac{\#\Delta' s\, on\, A}{d_A C_2} \tag{15}$$

$$E(cc) = \frac{E(\#\Delta' s\, on\, A)}{d_A C_2} \tag{16}$$

Let A have m neighbors. The probability of an edge between 2 vertices from the m neighbors is given by p.

$$E(\#\Delta' s\, on\, A) = mC_2 p \tag{17}$$

$$E(cc) = \frac{mC_2 p}{mC_2} = p \tag{18}$$

The expected value of the clustering coefficient is solely dependent on the probability of the edge needed to form the triangle.

Problem 5. For the graph in Figure 2 determine:

1. What is the minimum degree for a node to be considered a heavy hitter?
2. Which nodes are heavy hitters?

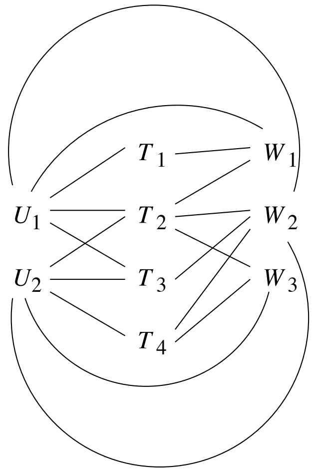

Figure 2: Tripartite Graph

3. Which triangles are heavy-hitter triangles?

Solution:

1. Total number of edges is 17. Therefore the minimum degree of a node to be a heavy hitter is $\sqrt{17}$ which is approximately 4.1. The minimum degree needed is 5.
2. Heavy hitter nodes are $U_1\, U_2\, T_2\, W_2$
3. Heavy hitter triangles are $< U_1, T_2, W_2 >$ and $< U_2, T_2, W_2 >$

Problem 6.

1. Extend the parallel algorithm discussed in the lecture to detect squares. That is for nodes a, b, c, d the edges $(a, b), (b, c), (c, d), (a, d)$ should exist in the graph. Write the pseudo-code map and reduce steps involved.
2. Does your proposed algorithm be extended for arbitrary sized polygons ?
3. Are there computational bottlenecks in the algorithm when there is skew (a power law distribution on the outdegrees) ? Outline rough ideas to overcome them (if at all).

Solution:
First number the vertices in the graph and use the rule that for an edge $< a, b >$, $a < b$. This is to avoid duplicate edges $< a, b >, < b, a >$
First detect all triangles using the algorithm described in the lecture. Instead of checking if the edge between u, w exists in E, assume that it does. Note that a square is made up of 2 triangles that share an edge.
Now for the next map-reduce job your input is the set of all triangles detected: $< a, b, c >, < a, b, d >, \ldots$
Map: emit an edge and the corresponding triangle. For the triangle $< a, b, c >$, emit $< (a, b), abc >$, $< (b, c), abc >$ and $< (a, c), abc >$.

Reducer: A single reducer gets all triangles that share an edge. Input to the reducer is $< (a, b), [abd, abc, \ldots] >$. Every possible pair of triangles in the list of values will give you a square. Output nC_2 squares where n is the number of triangles sharing that edge.
This solution is extensible for arbitrary polygons. An additional map reduce job is needed to combine triangles or squares to form a particular polygon. For example a pentagon can be formed by checking for a square first and then checking if another triangle shares an edge with the square.
The main bottleneck is the sheer number of triangles that can be generated.