Community Detection in Graphs: Finding overlaps

CS246: Mining Massive Datasets
Jure Leskovec, Stanford University
http://cs246.stanford.edu
Can we identify social communities?

Nodes: Facebook Users
Edges: Friendships
Facebook Network

Social communities

- High school
- Stanford (Squash)
- Stanford (Basketball)
- Summer internship

Nodes: Facebook Users
Edges: Friendships
Protein-Protein Interactions

Can we identify functional modules?

Nodes: Proteins
Edges: Physical interactions
Protein-Protein Interactions

Nodes: Proteins
Edges: Physical interactions

Functional modules
Non-overlapping vs. overlapping communities

Previous lecture

Today
Non-overlapping Communities

Finding good “cuts”
Communities as Tiles!

What if communities overlap?

Communities as “tiles”
(1) Given a model, we generate the network:

(2) Given a network, find the “best” model
Goal: Define a model for generating networks

- The model will have a set of “parameters” that we will later want to estimate (to detect communities)

Q: Given a set of nodes and their community memberships, how do communities “generate” edges of the network?
Generative model $B(V, C, M, \{p_c\})$ for graphs:
- Nodes $V$, Communities $C$, Memberships $M$
- Each community $A$ has a single probability $p_A$

(Later we fit the model to networks to detect communities, that is, for each node find communities it belongs to)
AGM: Generative Process

AGM generates the links:
For each pair of nodes in community $A$, we connect them independently with prob. $p_A$

- The overall edge probability is:

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

If $u, v$ share no communities: $P(u, v) = \varepsilon$

Think of this as an “OR” function: If at least 1 community says “YES” we create an edge
Recap: AGM networks
AGM can express a variety of community structures:
Non-overlapping, Overlapping, Nested
How do we detect communities with AGM?
Detecting communities with AGM:

Given a Graph $G(V,E)$, find the AGM

1) Affiliation graph $M$
2) Number of communities $C$
3) Parameters $p_c$
Maximum Likelihood Principle (MLE):

- **Given:** Data \( X \)
- **Assumption:** Data is generated by some model \( f(\Theta) \)
  - \( f \) ... model
  - \( \Theta \) ... model parameters
- **Want to estimate** \( P_f(X|\Theta) \):
  - The probability that our model \( f \) (with parameters \( \Theta \)) generated the data \( X \)
- **Now let’s find the most likely model that could have generated the data:** \( \arg \max_{\Theta} P_f(X|\Theta) \)
MLE for Graphs

How do we do MLE for graphs?

- **AGM generates a probabilistic adjacency matrix**
- We then flip all the entries of the probabilistic matrix to obtain the adjacency matrix of the graph $G$

For every pair of nodes $u, v$ AGM gives the prob. $p_{uv}$ of them being linked

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>0.10</th>
<th>0.10</th>
<th>0.04</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.10</td>
<td>0</td>
<td>0.02</td>
<td>0.06</td>
</tr>
<tr>
<td>0.10</td>
<td>0</td>
<td>0.04</td>
<td>0.06</td>
<td>0</td>
</tr>
<tr>
<td>0.04</td>
<td>0.06</td>
<td>0.06</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Flip biased coins

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- **The likelihood of AGM generating graph $G$:**

$$P(G \mid \Theta) = \prod_{(u,v)\in E} P(u, v) \prod_{(u,v)\notin E} (1 - P(u, v))$$
Given graph $G(V,E)$ and $\Theta$, we calculate likelihood that $\Theta$ generated $G$: $P(G|\Theta)$

$$P(G | \Theta) = \prod_{(u,v) \in E} P(u,v) \prod_{(u,v) \notin E} (1 - P(u,v))$$
Example: Likelihood of a Graph

Technically this is $1 - (1-p_C)$.

\[ p_{12} = p_{13} = p_C \]
\[ p_{23} = 1 - (1-p_C)(1-p_D) \]
\[ p_{24} = p_{34} = p_D \]
\[ p_{14} = \varepsilon \]

Likelihood of this graph =
\[ p_{12} p_{13} p_{23} p_{24} (1-p_{14}) (1-p_{34}) = (p_C)^2 (1 - (1-p_C)(1-p_D)) p_D (1-\varepsilon) (1-p_D) \]

Very close to 1; drop this factor
Our goal: Find $\Theta = (V, C, M, \{p_C\})$ such that:

$$
\arg \max_{\Theta} P(G | \Theta) = \prod_{u,v} P(u, v)^{G_{uv}} (1 - P(u, v))^{1 - G_{uv}}
$$

Often we take the logarithm of the likelihood, and call it log-likelihood: $l(\Theta) = \log P(G | \Theta)$

$$
l(G | \Theta) = \sum_{(u,v) \in E} \log(P(u, v)) + \sum_{(u,v) \notin E} \log(1 - P(u, v))
$$

$G_{uv}$ ... entry $(u,v)$ of adjacency matrix
Our goal is to find $B(V, C, M, \{p_C\})$ such that:

$$\arg \max_{B(V,C,M,\{p_C\})} \sum_{u,v \in E} \log P(u, v) + \sum_{u,v \notin E} \log(1 - P(u, v))$$

Problem: Finding $B$ means finding the bipartite affiliation network

- There is no nice way to do this
- Fitting $B(V, C, M, \{p_C\})$ is too hard, let’s change the model (so it is easier to fit)!
If $B(V, C, M)$ is given, finding $\{p_C\}$ is easy:

- Just write down the log-likelihood $l(G|\Theta)$ as a function of $\{p_C\}$ by computing each $P(u, v)$.
- Find $\{p_C\}$ that maximizes the log-likelihood.

\[
l(G|\Theta) = \sum_{(u,v) \in E} \log(P(u, v)) + \sum_{(u,v) \notin E} \log(1 - P(u, v))
\]
Relaxing AGM

- **Relaxation: Memberships have strengths**

- $F_{uA} \geq 0$: The membership strength of node $u$ to community $A$ (if $F_{uA} = 0$ then no membership)

- Each community $A$ links nodes independently:

  \[ P_A(u, v) = 1 - \exp(-F_{uA} \cdot F_{vA}) \]
Community membership strength matrix \( F \)

\[
\begin{array}{c}
\text{Communities} \\
\hline
\text{Nodes} \\
\end{array}
\]

- Probability of connection is proportional to the product of strengths
  - **Notice:** If one of the nodes doesn’t belong to the community \( A \) \( (F_{uA} = 0) \) then \( P_A(u, v) = 0 \)

- Prob. that **at least one** common community \( C \) links the two nodes:
  - \( P(u, v) = 1 - \prod_C (1 - P_C(u, v)) \)
Relaxing AGM

- Community $A$ links nodes $u, v$ independently:
  \[
P_A(u, v) = 1 - \exp(-F_{uA} \cdot F_{vA})
  \]

- Then prob. at least one common $C$ links them:
  \[
P(u, v) = 1 - \prod_C(1 - P_C(u, v))
  = 1 - \exp(-\sum_C F_{uC} \cdot F_{vC})
  = 1 - \exp(-F_u \cdot F_v^T)
  \]

- Example $F$ matrix:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1.2</th>
<th>0</th>
<th>0.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F_u$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>0.5</th>
<th>0</th>
<th>0</th>
<th>0.8</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F_v$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1.8</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F_w$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Node community membership strengths

Then: $F_u \cdot F_v^T = 0.16$

And: $P(u, v) = 1 - \exp(-0.16) = 0.14$

But: $P(u, w) = 0.88$

$P(v, w) = \varepsilon$

We assume that if $P$(edge)$=0$, then $P$(edge)$= \varepsilon$
How to find $F$

- **Task:** Given a network $G(V, E)$, estimate $F$
  - Find $F$ that maximizes the log-likelihood $l(F)$:
    \[
    \arg\max_F \sum_{u,v \in E} \log P(u,v) + \sum_{u,v \notin E} \log(1 - P(u,v))
    \]
    
    - where: $P(u,v) = 1 - \exp(-F_u \cdot F_v^T)$

- **Goal:** Find $F$ that maximizes $l(F)$:
    \[
    l(F) = \sum_{(u,v) \in E} \log(1 - \exp(-F_u F_v^T)) - \sum_{(u,v) \notin E} F_u F_v^T
    \]
Non-negative matrix factorization:

- Use a gradient based approach!
- But updating the whole $F$ takes too long
  - Computing the gradient takes quadratic time!
  - Our network are far too big to allow for that:
    - Network with 1M nodes, can have $10^{12}$ different edges
- Instead, update $F$ in small "units":
  - Update $F_{uC}$ for node $u$ while fixing the memberships of all other nodes
Optimizing Log-Likelihood

\[ l(F) = \sum_{u,v \in E} \log P(u, v) + \sum_{u,v \notin E} \log(1 - P(u, v)) \]

- Can rewrite \( l(F) = \frac{1}{2} \sum_{u \in V} l(F_u) \) where

\[ l(F_u) = \sum_{v \in \mathcal{N}(u)} \log(P(u, v)) + \sum_{v \notin \mathcal{N}(u)} \log(1 - P(u, v)) \]

Summing over all the edges is equivalent to summing over all the nodes and then over the neighbors \( \mathcal{N} \) of each node. \( \frac{1}{2} \) is since we count every edge twice.

\[ l(F_u) = \sum_{v \in \mathcal{N}(u)} \log(1 - \exp(-F_u F_v^T)) - \sum_{v \notin \mathcal{N}(u)} F_u F_v^T \]

\( \mathcal{N}(u) \).. Set out-outgoing neighbors
Optimizing Log-Likelihood

\[ l(F_u) = \sum_{v \in N(u)} \log(1 - \exp(-F_u F_v^T)) - \sum_{v \notin N(u)} F_u F_v^T \]

- Compute gradient of a single row \( F_u \) of \( F \):

\[ \nabla l(F_u) = \sum_{v \in N(u)} F_v \frac{\exp(-F_u F_v^T)}{1 - \exp(-F_u F_v^T)} - \sum_{v \notin N(u)} F_v \]

- Coordinate gradient ascent:
  - Iterate over the rows of \( F \):
    - Compute gradient \( \nabla l(F_u) \) of row \( u \) (while keeping others fixed)
    - Update the row \( F_u \): \( F_u \leftarrow F_u + \eta \nabla l(F_u) \)
    - Project \( F_u \) back to a non-negative vector: If \( F_{uC} < 0 \): \( F_{uC} = 0 \)
  - This is slow! Computing \( \nabla l(F_u) \) takes linear time!
However, we notice:

\[ \sum_{v \notin \mathcal{N}(u)} F_v = (\sum_{v} F_v - F_u - \sum_{v \in \mathcal{N}(u)} F_v) \]

- We cache \( \sum_{v} F_v \)
  - Note \( \sum_{v} F_v \) changes during each gradient descent step. But we cache it and update it only every so often (say every N steps).
- So, computing \( \sum_{v \notin \mathcal{N}(u)} F_v \) now takes linear time in the degree \( |\mathcal{N}(u)| \) of node \( u \)
  - In networks degree of a node is much smaller to the total number of nodes in the network, so this is a significant speedup!
Node Classification in Networks: Guilt by Association
Finding “Guilty Associates”

- Predict gene functions by guilty-by-association:

  ![Graph](image)

  - Red: Genes involved in protein folding
  - White: Genes with unknown function

- **Question:** Which additional genes are involved in “protein folding”? 
“Guilty Associates” Problem

- Let $W$ be a $n \times n$ (weighted) adjacency matrix over $n$ genes
- Let $y = \{-1, 0, 1\}^n$ be a vector of labels:
  - 1: positive gene, known to be involved in a gene function/biological process
  - -1: negative gene
  - 0: unlabeled gene
- Goal: Predict which unlabeled genes are likely positive
“Guilty Associates” Problem

- **Approach:** Learn a vector of discriminant scores $f$, where $f_i$ is **likelihood** that node $i$ is positive

- **Example:**

\[
y = [1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
\]

$W = \text{(weighted) adjacency matrix}$

$\mathbf{f} = ?$
Three approaches to Guilt by Association

1) Neighbor scoring
2) Using Random Walks
3) Label propagation
Approach 1: Neighbor Scoring

- Node score $f_i$ is weighted sum of the labels of $i$’s direct neighbors:

$$f_i = \sum_{j=1}^{n} W_{ij} y_j$$

- Example:

- $f_{GA} = W_{GA,MCA1} \cdot y_{MCA1}$
- $f_{GB} = W_{GB,CDC48} \cdot y_{CDC48} + W_{GB,TDH2} \cdot y_{CDC48}$
- $f_{GC} = W_{GC,TDH2} \cdot y_{TDH2}$

Green: Positive nodes
White: $f_i = 0$
Approach 1: Neighbor Scoring

- Node score $f_i$ is weighted sum of the labels of $i$’s direct neighbors:

$$f_i = \sum_{j=1}^{n} W_{ij} y_j$$

- Example:

- $f_{GA} = W_{GA,MCA1} \cdot y_{MCA1}$
- $f_{GB} = W_{GB,CDC48} \cdot y_{CDC48} + W_{GB,TDH2} \cdot y_{CDC48}$
- $f_{GC} = W_{GC,TDH2} \cdot y_{TDH2}$

- One half of $GC$’s neighbors are positives
- One third of $GA$’s neighbors are positives
- But: $f_{GC} = f_{GA}$ (if $W$ is binary)
Weighted Neighbors

- **Normalize matrix** $W$ by node degrees:

$$ f_i = \frac{1}{d_i} \sum_{j=1}^{n} W_{ij} y_j, \quad d_i = \sum_j W_{ij} $$

Matrix notation:

$$ f_i = D^{-1} W y $$

$$ D = \text{diag}(d) $$

- **Example:**

$$ f_{GA} = \frac{1}{3} W_{GA,MCA1} \cdot y_{MCA1} $$

$$ f_{GB} = \frac{1}{3} (W_{GB,CDC48} \cdot y_{CDC48} + W_{GB,TDH2} \cdot y_{TDH2}) $$

$$ f_{GC} = \frac{1}{2} W_{GC,TDH2} \cdot y_{TDH2} $$

Red: Positive genes
White: $f_i = 0$
Matrix $P = D^{-1}W$ is known as **Markov transition matrix**

- $D$ is a diagonal matrix with diagonal elements $d_i$
- $P$ is a **row stochastic matrix**, $\sum_j P_{ij} = 1$

Row $i$ is a probability distribution over the transition of a **random walk** starting at node $i$

$P_{ij}$ is probability of a **random walker following a link from node $i$ to node $j$**
**Approach 2: Random Walks**

- **Random walk** interpretation extends a direct neighbor approach to include **indirect neighbors**.

- **Idea:** Extend **direct neighbor scoring formula**  
  \[ f = D^{-1}Wy = Py \]  
  to include **second-degree neighbors**.

- **Probability of a random walk of length two** between node \( i \) and node \( j \) is:

  \[ \left[ P^2 \right]_{ij} = \sum_{k=1}^{n} P_{ik}P_{kj} \]
Consider **second-degree neighbors** when calculating node score $f_i$ as:

$$f_i = \sum_{j=1}^{n} P_{ij} y_j + \sum_{j=1}^{n} [P^2]_{ij} y_j$$
Example: Indirect Neighbors

\[ P = D^{-1}W \]

\[ f_i = \sum_{j=1}^{n} P_{ij} y_j + \sum_{j=1}^{n} [P^2]_{ij} y_j \]

- Direct neighbor of a positive gene
- Second-order neighbor of a positive gene

Red: Positive genes
White: \( f_i = 0 \)

\[ [P^2]_{ij} > 0 \] if there is a walk of length 2 between \( i \) and \( j \)

\[ f_{GA} = P_{GA,MCA1} \cdot y_{MCA1} \]

\[ f_{GE} = P_{GE,MCA1}^2 \cdot y_{MCA1} + P_{GE,TDH2}^2 \cdot y_{TDH2} + P_{GE,CDC48}^2 \cdot y_{CDC48} \]
This approach can be extended to include other nodes at a distance of length $r$ (usually $r < 4$)

Increasing $r$ beyond 2 often results in degradation of prediction performance

**Note:** Probability of a random walk from $i$ to $j$ in $r$ steps is given by $[P^r]_{ij}$

**Next:** Use random walks to derive label propagation
Approach 3: Label Propagation

- Label propagation generalizes neighborhood-based approaches by considering random walks of all lengths between nodes.

- The algorithm can be derived as:
  1. Iterative diffusion process [Zhou et al., NIPS 2004]
  2. Solution to a specific convex optimization task [Zhou et al., NIPS 2004, Zhu et al., ICML 2003]
  3. Maximum a posteriori (MAP) estimation in Gaussian Markov Random Fields [Rue and Held, Chapman & Hall, 2005]

- Next: Derivation based on a diffusion process.
Label Propagation: Intuition

Intuition: **Diffuse labels through edges of the network**

![Diagram](image)

(a) Initial Labels

- **Red:** positive nodes
- **White:** unlabeled nodes

(b) First Iteration

- **Red:** positive nodes
- **Pink:** $f_i > 0$
- **White:** $f_i = 0$
Diffusion Process: Idea

- **The diffusion process** is defined as an **iterative process**
- **Diffuse labels through network edges:**
  - Start with initial label information, $f^{(0)}_i = y_i$
  - In each iteration, each node $i$ receives **label information from $i$’s neighbors**, and also **retains its initial label**
  - $\lambda$ specifies **relative amount** of label information from $i$’s neighbors and its initial label
  - Finally, the label of each unlabeled node is set to be the class (-1 or 1) of which it has **received most information**
The diffusion process is defined as the following iteration:

At iteration $r = 0$, define $f_i^{(0)} \leftarrow y_i$

At iteration $r + 1$, the score of node $i$ is the weighted average of the scores of $i$'s neighbors in iteration $r$, and $i$'s initial label:

$$ f_i^{(r+1)} \leftarrow (1 - \lambda)y_i + \lambda \sum_{j=1}^{n} W_{ij} f_j^{(r)} $$

$0 < \lambda < 1$ is model parameter.
Diffusion Process: Intuition

\[ f_i^{(r+1)} \leftarrow (1 - \lambda) y_i + \lambda \sum_{j=1}^{n} W_{ij} f_j^{(r)} \]

- ⇒ Discriminant scores \( f \) are weighted sum of walks of all lengths between the nodes

- ⇒ High score \( f_i \) is assigned to \( i \) if \( i \) is connected to positively labeled nodes with many short walks
Diffusion Process: Intuition

- \( f^{(R)} \) can be rewritten as:
  \[
  f^{(R)} = (1 - \lambda) \sum_{r=0}^{R} (\lambda W)^r y
  \]
  - \([W^r]_{ij} > 0\) if a walk of length \( r \) between \( i \) and \( j \)
  - Weight \( \lambda^r \) decreases with increasing distance

- \( \Rightarrow \) Discriminant scores \( f \) are \textit{weighted sum of walks of all lengths} between the nodes

- \( \Rightarrow \) \textit{High score} \( f_i \) is assigned to \( i \) if \( i \) is connected to positively labeled nodes with \textit{many short walks}
Diffusion Process: Example

(a) Initial Labels

\[ f^{(0)} = y \]

(b) First Iteration

\[ f^{(1)} = \lambda W y + (1 - \lambda) y \]

(c) Second Iteration

All nodes reachable with a walk of length 2 are assigned a non-zero value

\[ f^{(2)} = \lambda W f^{(1)} + (1 - \lambda) y \]

Red: positive nodes
Pink: \( f_i > 0 \)
White: \( f_i = 0 \)
Diffusion Process: Example

(a) Initial Labels
\[ f^{(0)} = y \]

(b) First Iteration
\[ f^{(1)} = \lambda Wy + (1 - \lambda)y \]

(c) Second Iteration
\[ f^{(2)} = \lambda Wf^{(1)} + (1 - \lambda)y \]

(d) Final Scores
\[ f = (1 - \lambda) \sum_{r=0}^{\infty} (\lambda W)^r y \]

All nodes reachable with a walk of length 2 are assigned a non-zero value.

Score
- high
- low

Red: positive nodes
Pink: \( f_i > 0 \)
White: \( f_i = 0 \)
Normalize Adjacency Matrix $W$

- **Problem:** The infinite sum does not converge in general

- **Solution:** Normalize $W$ before diffusion:
  - **Symmetric** normalization:
    \[ S = D^{-1/2}W D^{-1/2} \]
    
    - Signal is spread in a **breadth-first search** manner
  
  - **Asymmetric** normalization:
    \[ P = D^{-1}W \]

\[
f = (1 - \lambda) \sum_{r=0}^{\infty} (\lambda W)^r y
\]
Multi-label node classification:
Node (gene) has 0+ labels (functions):

1. Observe a fraction of nodes and their labels
2. For each label, use a diffusion approach to predict node labels \( f \) of the remaining nodes

Select optimal value for \( \lambda \) using cross-validation