dh(x)=bc

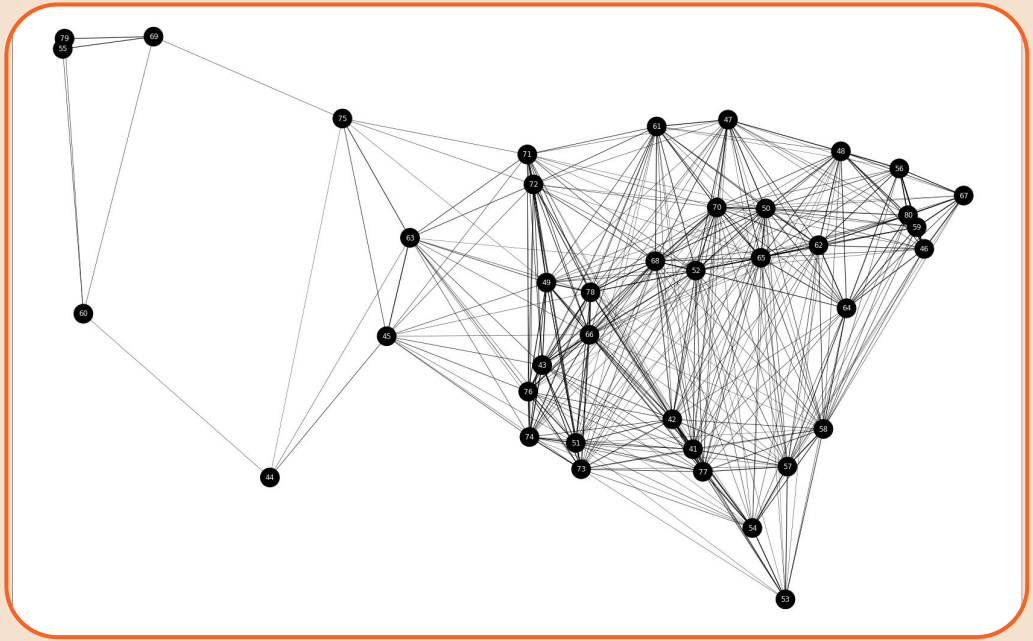
Random Graphs: *Block Partitions and Embeddings*

Team 2

Chenhongshu Yu, Adam Levav

Provided Data

- 40 cities (41~80) from SGB128 dataset
- 780 weighted edges
- We use Python for our project, using **networkx** package to load the data into matrices and graphs



I. Random Graph Model Testing

Clique Calculation & Estimation

- **Goal:** estimate the number of *k*-cliques under two different models of random graphs

- Sort edges descending by weight
- For each set of *k* largest edges, create a graph and compute:
  - Actual # of 3- and 4-cliques
  - Estimated # of 3- and 4-cliques under:
    - Erdos-Renyi (ER) random graph model
    - Symmetric Stochastic Block Model (SSBM)
- Find exponential fit for each; compare

Clique Calculation & Estimation

- Actual # of 3- and 4-cliques: We check all edges and nodes
- Estimated # of 3 - and 4-Cliques:

Erdös-Rényi (ER):

➢ MLE Estimator:

$$P_{MLE} = \frac{m}{\binom{n}{2}} = \frac{2m}{n(n-1)}$$

➢ Estimated # of q-cliques:

$$E[q-cliques] = \binom{n}{2} p^{\frac{q(q-1)}{2}}$$

Symmetric Stochastic Block Model (SSBM)

➢ a_MLE: $a_{MLE} = \frac{2(m_{11}+m_{22})}{n_1(n_1-1)+n_2(n_2-1)}$

➢ b_MLE: $b_{MLE} = \frac{m_{12}}{n_1 n_2}$

➢ a_MM & b_MM: $a_{MM} = b_{MM} = \frac{1}{2}(c_1 + \sqrt[3]{2c_2 - {c_1}^3})$

➢ Algo (1) & (2):

$$p = \frac{2m}{n(n-1)}, \delta = \frac{6t}{n(n-1)(n-2)} - p^3$$

$$A_1 = max(0, 2p-1), A_2 = min(1, 2p)$$

$$P(A_1) = (A_1) - p)^3 - \delta, P(A_2) = (A_2 - p)^3 - \delta.$$

➢ a_CMM & b_CMM: $a_{CMM} = p + \sqrt[3]{\delta}$, $b_{CMM} = p - \sqrt[3]{\delta}$

Clique Calculation & Estimation

- Actual # of 3- and 4-cliques: iterative approach
-

```python
def count_3_4_cliques(G, num_3_4_cliques, new_edge):
    a, b, *data = new_edge
    # 3 cliques
    adj_a = neighbors(G,a)
    adj_b = neighbors(G,b)
    all_c = adj_a & adj_b # intersection of neighbors
    num_3_4_cliques[3] += len(all_c)
    # 4 cliques
    all_d = [v for c in all_c for v in neighbors(G,c)
             if v in all_c]
    num_3_4_cliques[4] += len(all_d)//2

    # return
    G.add_edge(a, b)
    return G, num_3_4_cliques
```

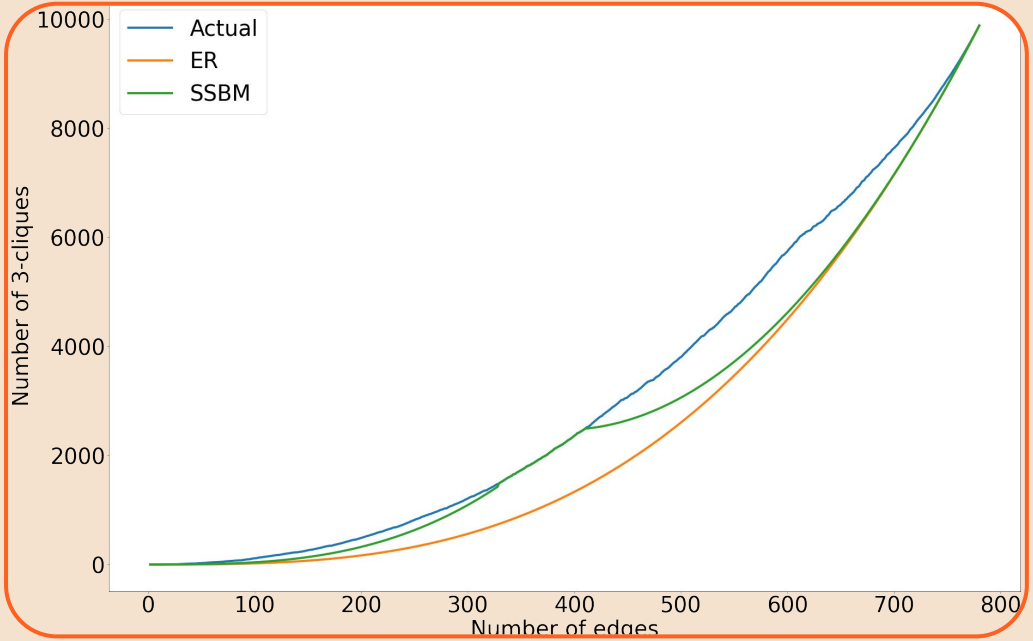
Appends the new edge to the previous edges and updates the prev_cliques structure accordingly.

For a new edge (a,b), check for all nodes c such that (a,b) and (a,c) are already in the graph, and append those to Counter.

Also check for all nodes c and d such that (a,c), (c,d), and (d,b) are in the graph, and append those to the new Counter.

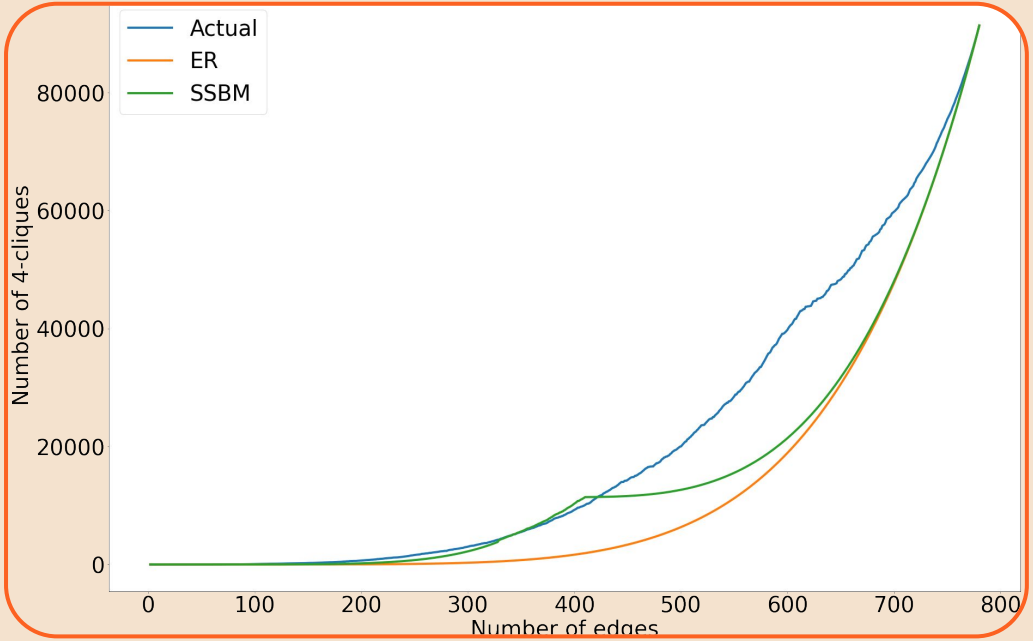
3-cliques

- ER under-estimates
- SSBM only fits well in specific region (350 - 450 nodes); we expect it to fit everywhere



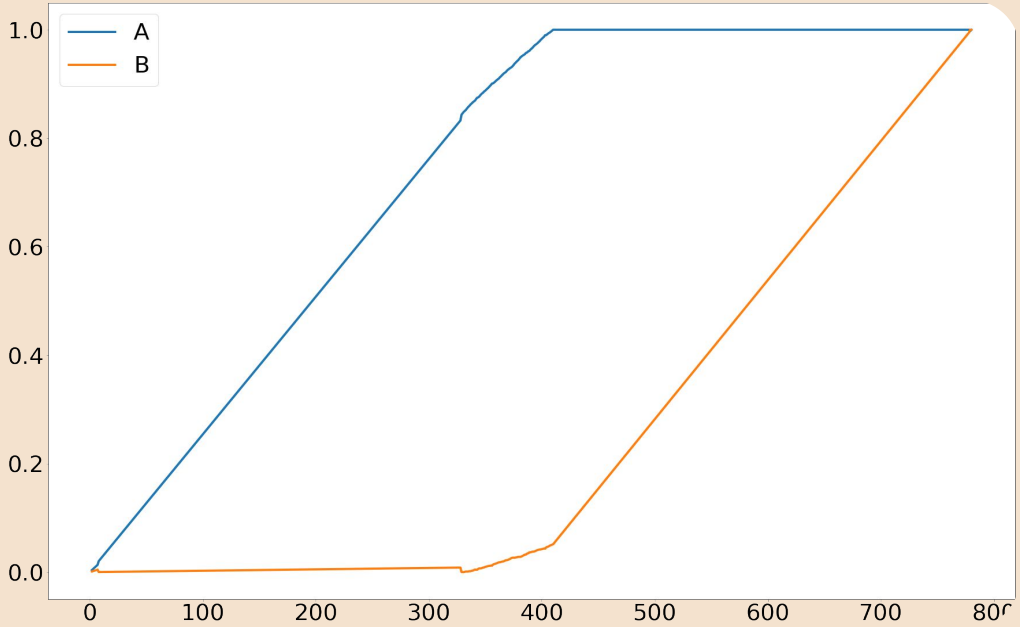
4-cliques

- ER under-estimates
- SSBM fits pretty well in the early stages (~410), but after that it fits dramatically bad with the actual 4-cliques



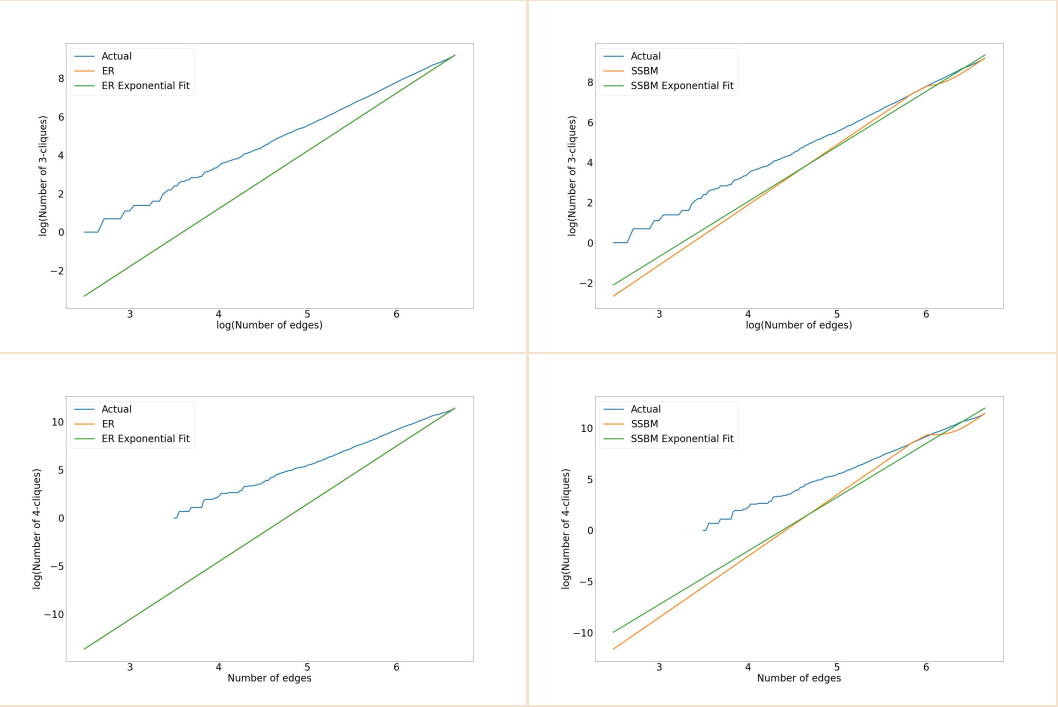
Estimated a & b for SSBM

- b near 0 before 325 edges
- a equals 1 after 410 edges
- Only in the middle do a & b have meaningful values



Exponential Fits

- ER exponential fit matches perfectly, as expected
- SSBM exponential fit with slight error, but overall fit matches well
- By looking at the graphs, SSBM model seems to fit slightly better



Comparing Approximations

- SSBM showed lower MSE for both 3- and 4-cliques, so it's the better approx.
- Skip first few edge sets b/c there are no 3-cliques / 4-cliques in those; log(0) is undefined

# cliques	ER	SSBM
3	$6.3 \cdot 10^5$	$1.2 \cdot 10^5$
4	$3.8 \cdot 10^8$	$4.0 \cdot 10^7$

Mean squared error of ER and SSBM *k*-clique estimates

II. Community Detection

Community Detection

- **Goal:** partition nodes into two communities using multiple methods
- Apply spectral methods using:
  - Weight matrix (W)
  - Laplacian matrix (Δ)
  - Normalized Laplacian matrix (Δ^{hat})
- Compute agreement between communities from these methods

Community Detection

- We use the networkx package to load our data into matrices and graphs
- Apply the different spectral algorithms with W, Δ, Δ^{hat}
- Compute the agreement matrix between these partitions

.	Weight	Laplacian	Normalized Laplacian
Weight	40	23	21
Laplacian	23	40	24
Normalized Laplacian	21	24	40

- Visualization for each algorithm
  - Draw edges according to the adjacency matrix A
  - Draw edges according to the weight matrix W

Spectral Algorithm with …

- With Weight (W):
  - Compute the second largest eigenpair of A : (f_2, μ_2), with $Wf_2 = \mu_2 f_2$.
  - Define the partition $\Omega_1 = \{k : f_2(k) > 0\}$, $\Omega_2 = \{k : f_2(k) \leq 0\}$. Set d = 2.
- With Weighted Laplacian (Δ):
  - Compute the weighted graph Laplacian $\Delta = D - A$, with $D = \mathrm{Diag}(A \cdot 1)$.
  - Compute the second smallest eigenpair: (e_1, λ_1), with $\Delta e_1 = \lambda_1 e_1$ and $\lambda_1 > 0 = \lambda_0$.
  - Define the partition $\Omega_1 = \{k : e_1(k) > 0\}$, $\Omega_2 = \{k : e_1(k) \leq 0\}$. Set d = 2.
- With Normalized Weighted Laplacian (Δ^{hat}):
  - Compute the symmetric normalized weighted graph Laplacian $\Delta^{hat} = I - D^{-1/2}AD^{-1/2}$, with $D = \mathrm{Diag}(A \cdot 1)$.
  - Compute the second smallest eigenpair: (e_1, λ_1), with $\Delta^{hat} e_1 = \lambda_1 e_1$ and $\lambda_1 > 0 = \lambda_0$.
  - Define the partition $\Omega_1 = \{k : e_1(k) > 0\}$, $\Omega_2 = \{k : e_1(k) \leq 0\}$. Set d = 2.

Partitions

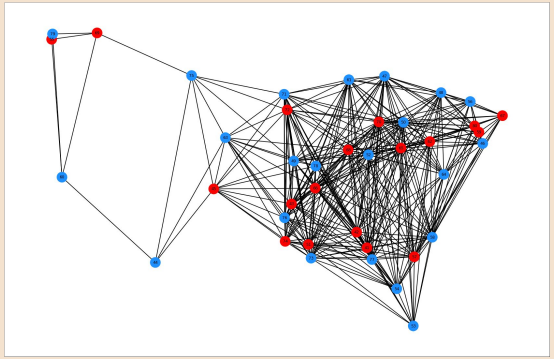
Weight	{0, 1, 2, 4, 7, 8, 9, 39, 11, 12, 13, 15, 17, 21, 23, 25, 26, 28}	{3, 5, 6, 10, 14, 16, 18, 19, 20, 22, 24, 27, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38}
Laplacian	{32, 33, 2, 3, 4, 34, 37, 39, 9, 10, 11, 12, 16, 17, 18, 24, 25, 28}	{0, 1, 5, 6, 7, 8, 13, 14, 15, 19, 20, 21, 22, 23, 26, 27, 29, 30, 31, 35, 36, 38}
Norm. Laplacian	{0, 2, 34, 36, 5, 37, 7, 38, 11, 14, 15, 16, 17, 19, 20, 21, 23, 30}	{1, 3, 4, 6, 8, 9, 10, 12, 13, 18, 22, 24, 25, 26, 27, 28, 29, 31, 32, 33, 35, 39}

Agreement Matrix

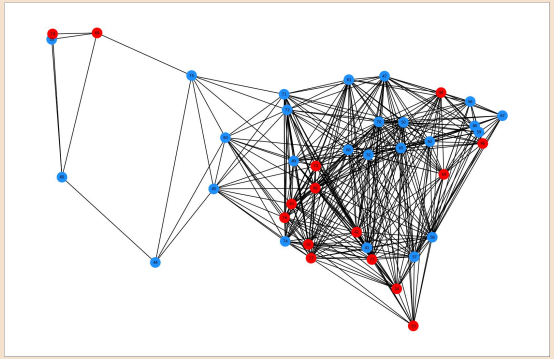
- Quite low partition agreement between different methods (slightly above random chance)

	Weight	Laplacian	Norm. Laplacian
Weight	100%	57.5%	52.5%
Laplacian	57.5%	100%	60%
Norm. Laplacian	52.5%	60%	100%

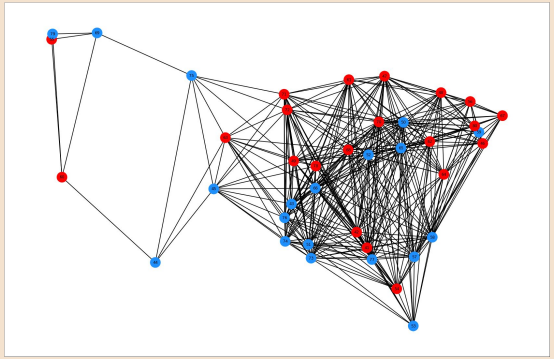
Visualization
— Adjacency Matrix A



Weighted

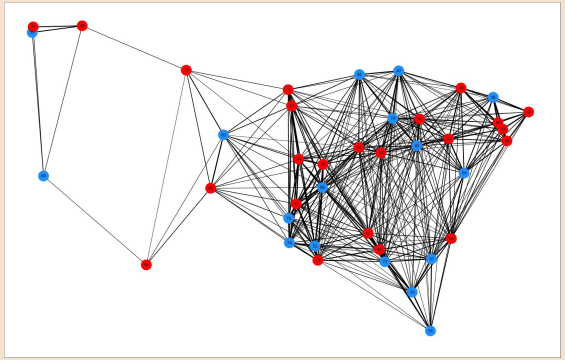


Normalized Laplacian

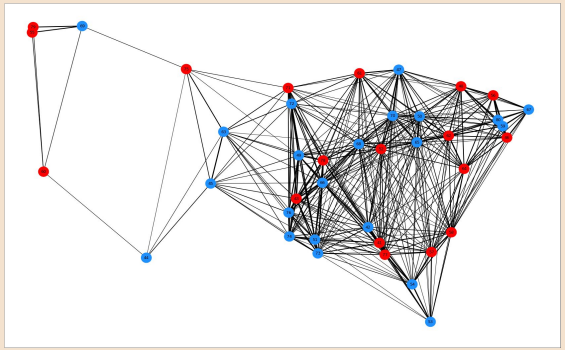


Laplacian

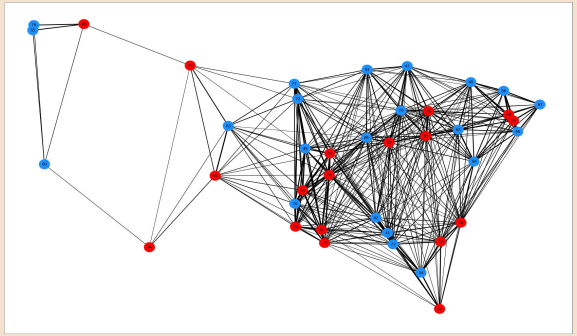
Visualization
— Weight Matrix W



Weighted



Normalized
Laplacian



Laplacian

III. Data Embedding

Data Embedding

- **Goal:** Find a lower-dimensional representation of the graph using multiple methods

- Apply two embedding algorithms:

1. Laplacian Eigenmap data embedding, target dimension 2
2. LLE dimension reduction after Laplacian Eigenmap data embedding:
    - Laplacian Eigenmap data embedding algorithm, N = 10
    - LLE algorithm with non-negativity constraints on previous step (d = 2; K = 4)

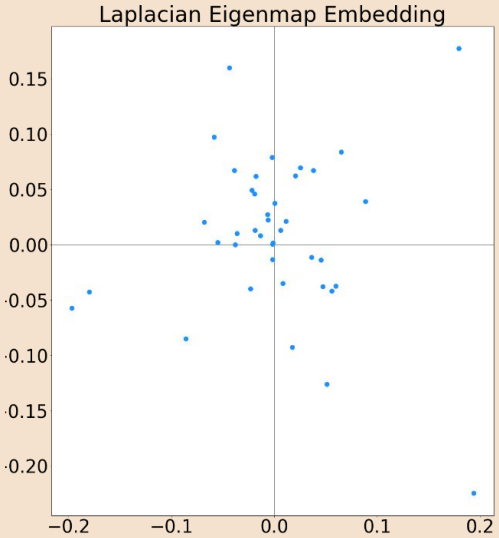
Laplacian Eigenmap

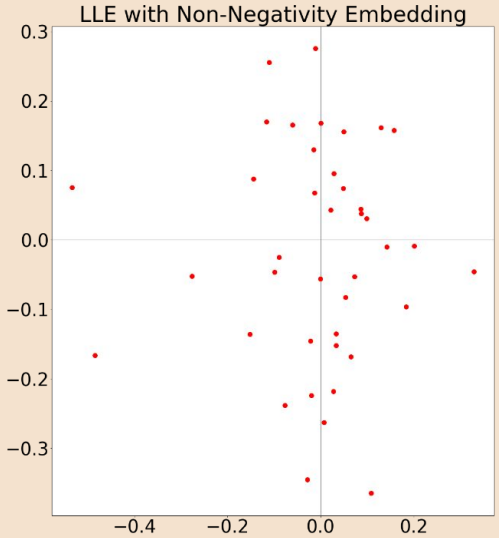
1) Construct the diagonal matrix $D = \mathrm{diag}(D_{ii})_{1 \le i \le n}$, where $D_{ii} = \sum^{n}_{k=1} W_{i,k}$
2) Construct the normalized Laplacian $\Delta^{hat} = I - D^{-1/2} W D^{-1/2}$.
3) Compute the bottom $d + 1$ eigenvectors $e_1, \cdots, e_{d+1}$, $\Delta^{hat} e_k = \lambda_k e_k$, $0 = \lambda_1 \le \cdots \le \lambda_{d+1}$.
4) Construct the $d \times n$ matrix Y, $Y = [e_2^T \ldots e_{d+1}^T]^T D^{-1/2}$
5) The new geometric graph is obtained by converting the columns of Y into n d -dimensional vectors: $[\, y_1 \cdots y_n \,] = Y$

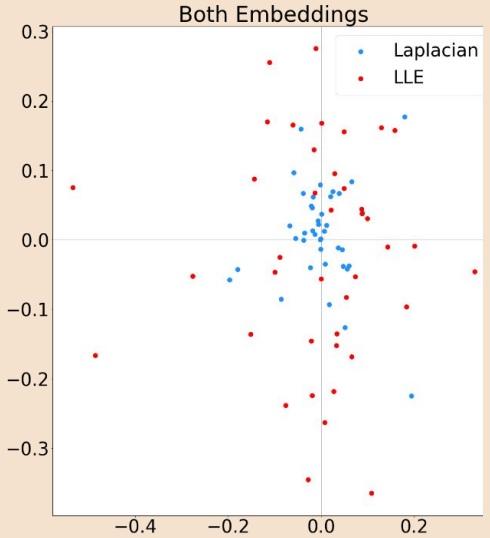
Local Linear Embedding (LLE)

1) Finding the weight matrix B: For each point i do the following:
   a) Find its closest K neighbors, say V_i ; Let $r : V_i \longrightarrow \{1, 2, \cdots, K\}$ denote indexing map
   b) Compute the K × K local covariance matrix C, $C_{r(j),r(k)} = \langle x_j - x_i , x_k - x_i \rangle$
   c) Solve for u, minimize $u^T Cu$, subject to $u \geq 0$, $u^T \cdot 1 = 1$ where 1 denotes the K-vector of 1's.
   d) Set $B_{i,j} = u_{r(j)}$ for $j \in V_i$.
2) Solving the Eigen Problem:
   a) Create the (typically sparse) matrix $L = (I - B)^T(I - B)$;
   b) Find the bottom d + 1 eigenvectors of L (the bottom eigenvector would be $[1, \cdots, 1]^T$ associated to eigenvalue 0) $\{e_1, e_2, \cdots, e_{d+1}\}$;
   c) Discard the last vector (the constant eigenvector) and insert all other eigenvectors as rows into matrix Y, $Y = [e_2^T \ldots e_{d+1}^T]^T$

Embeddings


Laplacian Eigenmap Embedding
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
−0.20
−0.2
−0.1
0.0
0.1
0.2


LLE with Non-Negativity Embedding
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
−0.4
−0.2
0.0
0.2


Both Embeddings
Laplacian
LLE
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
−0.4
−0.2
0.0
0.2

Issues Encountered

- Part I
  - Calculating the 3- and 4- cliques: naive approach would take days to finish; ended up using iterative method
  - Got complex estimated values for cliques when *k* is low
  - Aforementioned weirdness with SSBM
- Part II
  - Possible errors in implementations
- Part III
  - Hard to implement algorithms properly; making sure dimensions align properly

Thanks !

Do you have any questions?

Part 1 Code & Graphs: Adam
Part 2 Code & Graphs: Adam
Part 3 Code & Graphs: Adam
Slides: Chenhongshu, Adam