Network Cross-Validation

Aaron, Anthony

Department of Mathematics and Statistics

Washington University in St. Louis

January 17, 2024

Overview

Introduction and Background

Cross Validation and Model Selection

- **Purpose**: Evaluate model generalization.
- **Method**:
    - Split data into k folds.
    - Train on $k - 1$ folds, test on 1 fold.
    - Repeat for each fold, average results.
- **Types**:
    - k-Fold (common, $k = 5 - 10$)
    - Leave-One-Out (for small datasets)
- **Advantage**: Reduces performance estimation bias.

Network Cross-Validation for Determining the Number of Communities in Network Data

Network Cross-Validation

- A substantial amount of work has been done on the community recovery problem
- Determine the number of communities K in a network remains a challenge
- Perform Cross-Validation on graph $\hat{K}$

Block-wise Node-Pair Splitting

Fitting and Testing set Construction

In order to perform Cross-validation, we need to first divide the nodes into $(\mathcal{N}_1, \mathcal{N}_2)$ for training and testing purposes. Unlike the traditional method, which only divides the node randomly into two sets $(\mathcal{N}_1, \mathcal{N}_2)$, as long as $|\mathcal{N}_1| = V \cdot |\mathcal{N}_2|$ the method proposed in this paper has the following advantages

- Membership of all nodes can be inferred from the fitting sets.
- Fitting and testing sets remained independent.

Block-wise Node-Pair Splitting Continued

Overview

- Node pairs (i, j) for $i \in N_1$ and $j \in N_1 \cup N_2$ form the fitting set.
- Node pairs (i, j) with $i, j \in N_2$ are used as the testing set.
- This method ensures comprehensive relationship modeling and independent edge formation analysis.

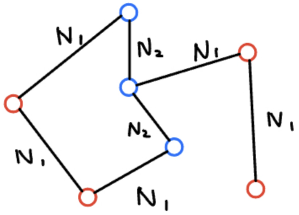


Figure 1: Illustration of Block-wise Node-Pair Splitting: Nodes are divided into subsets N_1 (red) and N_2 (blue).

Estimating parameters from the rectangular matrix

Estimation

Given $(\mathcal{N}_1, \mathcal{N}_2)$, we want to learn the model parameters $(\hat{g}, \hat{B})$, where $\hat{B} = [0,1]^{K \times K}$ is the symmetric matrix representing the community-wise edge probability, and $\hat{g} = \{1, 2, ..., K\}^n$ is the community membership vector, from $A^{(1)} = (A^{(11)}, A^{(12)})$, which is the adjacency matrices for nodes in $(\mathcal{N}_1, \mathcal{N}_2)$.

$$A = \begin{pmatrix} A^{(11)} & A^{(12)} \\ A^{(21)} & A^{(22)} \end{pmatrix}$$

Validation using $\mathcal{N}_2$

Input: A set $\mathcal{K}$ of candidate values for K, adjacency matrix A, number of folds $V \geq 2$.

- split the adjacency matrix into $V \times V$ blocks of equal size
- For $1 \leq \nu \leq V$, and for each $\tilde{K} \in \mathcal{K}$,
    - Estimate the model parameter $(\hat{g}^{(\nu)}, \hat{B}^{(\nu)})$ by removing the rows of A in the subset $\mathcal{N}_\nu$. (See Previous Slide)
    - Calculate the loss:

$$\hat{L}^{(\nu)}(A, \tilde{K}) = \sum_{i,j \in \mathcal{N}_\nu, i \neq j} (A_{ij} - \hat{P}_{ij}^{(\nu)})^2,$$

    where $\hat{P}_{ij}^{(\nu)} = \hat{B}^{(\nu)}_{\hat{g}_i^{(\nu)} \hat{g}_j^{(\nu)}}$.

    - Output

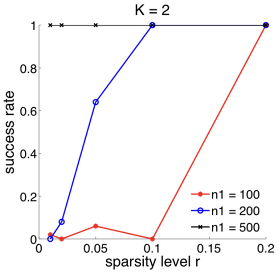
$$\hat{K} = \arg\min_{\tilde{K} \in \mathcal{K}} \sum_{\nu=1}^{V} \hat{L}^{(\nu)}(A, \tilde{K}).$$

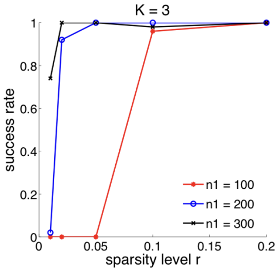
Recovering $\hat{K}$

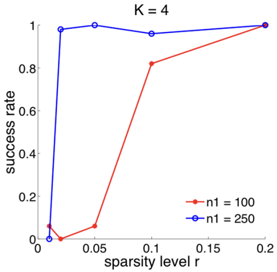
Number of communities

Note that the term $\sum_{v=1}^{V} \hat{L}^{(v)}(A, \tilde{K})$ is minimized when $\tilde{K} = K$. If $\tilde{K}$ is too small, the fitted model is insufficiently trained and **underfit**, whereas if $\tilde{K}$ is too big, the model will **overfit**.

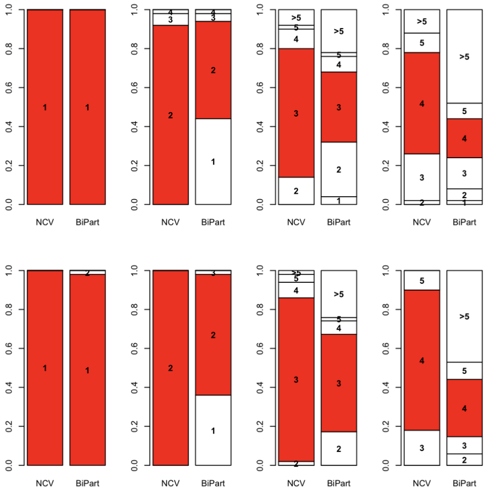
Simulation 1: Edge sparsity and Community Imbalance







Simulation 2: general block structures and comparison to recursive bipartition



Simulation 3: degree corrected block models

		SBM				*DCBM*			
		$K=1$	2	3	4	$K=1$	2	3	4
$n=300$	$\hat{P}(\hat{T}=T)$	1	1	1	1	1	0.68	0.44	0.42
	$\hat{P}(\hat{K}=K\|\hat{T}=T)$	1	1	0.98	0.92	1	0.41	0	0
$n=600$	$\hat{P}(\hat{T}=T)$	1	1	1	1	1	1	0.96	0.98
	$\hat{P}(\hat{K}=K\|\hat{T}=T)$	1	1	1	0.98	1	1	0.42	0
$n=1200$	$\hat{P}(\hat{T}=T)$	1	1	1	1	1	1	1	1
	$\hat{P}(\hat{K}=K\|\hat{T}=T)$	1	1	1	0.98	1	1	1	1

An Improvement: Network Cross-Validation by Edge Sampling

What's to consider in sampling?

Quality of data in a sample is fundamental for statistical analysis. We have seen a simple sampling in Chen and Lei (2015) and its performance in cross-validation and Li et al. (2020) proposed an improvement.

Setting:

- Nodes of network: $\mathcal{V} = \{1, 2, \cdots, n\} =: [n]$.
- $n \times n$ adjacency matrix: A.
- $A_{ij} = A_{ji} \sim Ber(p)$. $\mathbb{E}A = M$ is a matrix of probabilities.
- $G_{n,p}$ model: assign to each graph g with n nodes and m edges.

$$P(G = g) = p^m (1 - p)^{\binom{n}{2} - m}$$

The general network analysis task:

- estimate M from the data A, under various structural assumptions.

cross-validation on networks:

- how to treat the resulting partial data which is no longer a complete network.

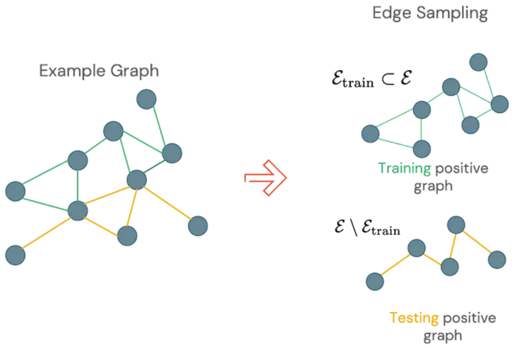
General Approach

Edge cross-validation (ECV) algorithm

(1) Obtain training set by splitting node pairs: edge sampling with probability of inclusion p instead of node sampling;

(2) obtain training data $\hat{A}$ and testing data by matrix completion

(3) evaluate models $1, \cdots, Q$


Example Graph
Edge Sampling
$\mathcal{E}_{\text{train}} \subset \mathcal{E}$
Training positive graph
$\mathcal{E} \setminus \mathcal{E}_{\text{train}}$
Testing positive graph

Source: https://medium.com/@ajmal.t.aziz/

ECV Algorithm I

Algorithm 1 (ECV Algorithm):
Input: an adjacency matrix A, a loss function L, a set $\mathcal{C}$ of Q candidate models, the training proportion p, and the number of replications N.

```
# 1. Select rank for matrix completion
K_hat = select_rank(method='Algorithm_2')

# 2. Main loop
for n in range(1, N+1):
   # (a) Choose subset of node pairs
   Omega = Bin(set=V⋆V, probability=p)

   # (b) Apply matrix completion
   A_hat = low_rank_matrix_completion(A, Omega, K_hat)

   # (c) Model fitting and loss evaluation
   for q in range(1, Q+1):
      fit_model_on(A_hat)
      loss_q_n = evaluate_loss(A, Omega_complement)

# 3. Determine the best model
L_q = calculate_average_loss(loss_q_n, N)
best_model = argmin(L_q)
```

ECV Algorithm II

Algorithm 2 (Rank Selection)
Rank selection is itself another parameter tuning/model selection. We need to choose a rank $\hat{K}$ so that $\hat{A}$ preserves structural information of A. Since this is a parameter tuning/model selection, we need to design a function `calculate_loss` for the rank selection algorithm. We can use the sum of squared errors on the held-out entries, $= \sum_{(i,j)\in\Omega^c} \left(A_{ij} - \hat{A}_{ij}\right)^2$, or, when A is binary, the binomial deviance as the loss function to optimize.

```
for n in range(1, N+1):
   Omega = Bin(set=V⋆V, probability=p)
   A_hat = low_rank_matrix_completion(A, Omega, K_hat)

   for k in range(1, K_max+1):
      A_hat = low_rank_matrix_completion(A, Omega, rank=k)
      loss_k_n = calculate_loss(A, Omega_complement)

# Determine the minimum loss rank
L_k = calculate_average_loss(loss_k_n, N)
optimal_rank = argmin(L_k)
```

Matrix Completion Process I

- P_Ω **Operator**: $P_\Omega : \mathbb{R}^{n\times n} \to \mathbb{R}^{n\times n}$; $P_\Omega(A)_{ij} = \begin{cases} A_{ij} & \text{if } (i,j) \in \Omega \\ 0 & \text{otherwise} \end{cases}$
- **Optimization Problem**:

$$\min_W \; F(P_\Omega(W), P_\Omega(A)), \text{ subject to } \operatorname{rank}(W) \le \hat{K}$$

Where W is the matrix to optimize, F is the loss function, and $\hat{K}$ is the rank constraint.

- **Solver - Singular Value Thresholding**:

$$\hat{A} = S_H\left(\frac{1}{p}P_\Omega(A), \hat{K}\right)$$

where $S_H\left(P_\Omega A, \hat{K}\right)$ is rank $\hat{K}$ truncated SVD of a matrix $P_\Omega A$. i.e., if the SVD of $P_\Omega A$ is $P_\Omega A = UDV^T$ where $D = \mathrm{diag}\left(\sigma_1, \cdots, \sigma_n\right), \sigma_1 \geq \cdots \sigma_n \geq 0$, then $S_H\left(P_\Omega A, \hat{K}\right) = UD_{\hat{K}}V^T$, where $D_{\hat{K}} = \mathrm{diag}\left(\sigma_1, \cdots, \sigma_{\hat{K}}, 0, \cdots, 0\right)$.

SVT algorithm approximates matrix A with a lower rank matrix $\hat{A}$, using rank threshold $\hat{K}$.

- **Rationale**:
  - Assumes low-rank approximation of the network matrix.
  - Applicable to various network types (directed/undirected, binary/weighted).
  - Stability selection incorporated for robustness.

Theoretical Justification I

Intuitively, ECV should work well if $\hat{A}$ reflects relevant structural properties of the true underlying model. The following theorem formalizes this intuition. All results will be expressed as a function of

- the number of nodes n;
- the sampling probability p which controls the size of the training set;
- the rank K of the true matrix M;
- an upper bound on the expected node degree d, defined to be any value satisfying $\max_{ij} M_{ij} \leq d/n$, a crucial quantity for network concentration results.

We can always trivially set $d = n$, but we will also consider the sparse networks case with $d = o(n)$.

Theorem

Let M be a probability matrix of rank K and d as defined above. Let A be an adjacency matrix with edges sampled independently and $\mathbb{E}(A) = M$. Let Ω be an index matrix for a set of node pairs selected independently with probability $p \geq C_1 \log n/n$ for some absolute constant C_1, with $\Omega_{ij} = 1$ if the node pair (i, j) are selected and 0 otherwise. If $d \geq C_2 \log(n)$ for some absolute constant C_2, then with probability at least $1 - 3n^{-\delta}$ for some $\delta > 0$, the completed matrix $\hat{A}$ defined in (2) with $\hat{K} = K$ satisfies

$$\|\hat{A} - M\| \leq \tilde{C} \max\left(\sqrt{\frac{Kd^2}{np}}, \sqrt{\frac{d}{p}}, \frac{\sqrt{\log n}}{p}\right)$$

where $\tilde{C} = \tilde{C}(\delta, C_1, C_2)$ is a constant that only depends on C_1, C_2 and δ. This also implies

$$\frac{\|\hat{A} - M\|_F^2}{n^2} \leq \frac{\tilde{C}^2}{2} \max\left(\frac{K^2 d^2}{n^3 p}, \frac{Kd}{n^2 p}, \frac{K \log n}{n^2 p^2}\right).$$

Theorem

Assume A is generated from a random dot product graph model satisfying 1 and 2 (referring to eigenvalue gap and Incoherent matrix assumption), with latent space dimension K. Let $\hat{K}$ be the output of Algorithm 2. If the sum of squared errors is used as the loss and the expected degree satisfies $\lambda_n / \left(n^{1/3} \log^{4/3} n \right) \to \infty$,

$$\mathbb{P}(\hat{K} < K) \to 0$$

Model Selection: Parametric and Nonparametirc Tuning in Graphon Estimation I

Algorithm 2 and Theorem 2 displayed before give the Model-free rank estimators that can be specifically applied to dot product graph model. The first part of our presentation also gives the implementation of cross validation on Stochastic Block Models (SBM). The random dot product graph model can include the stochastic block model as a special case, but only if the probability matrix M of the stochastic block model is positive semi-definite. We turn to the third type of low-rank applicable model: latent space and graphon.

Latent space model assumes the nodes correspond to n latent positions $Z_i \in \mathbb{R}^K$, and the probability matrix is some function of the latent positions, for example, the distance model $f\left(M_{ij}\right) = \alpha - \left\|Z_i - Z_j\right\|$, where f is a known function, such as the logit function.

More generally, Aldous-Hoover representation we've talked about in class says that the probability matrix of any exchangeable random graph can be written as $M_{ij} = W(U_i, U_j)$ for a symmetric function $W : [0,1] \times [0,1] \to [0,1]$, determined up to a measure-preserving transformation. W is called a **graphon**.

Parametric: Zhang et al. (2017) proposed a neighborhood smoothing estimation for a graphon model that depends on a tuning parameter h which controls the degree of smoothing. The theory suggests $h = \tau(\log n/n)^{1/2}$ for some τ.

Nonparametric: The sampling is also applied to a dataset compiled by Ji & Jin (2016). This dataset contains information (title, author, year, citations and DOI) about all papers published between 2003 and 2012 in four top statistics journals.

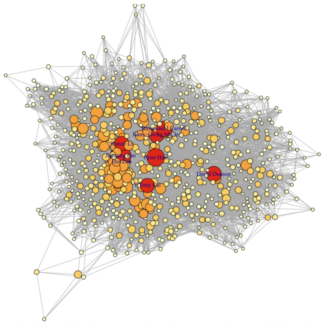


Fig. 2: The core of statistician citation network. The network has 706 nodes with node citation count (ignoring directions) ranging from 15 to 703. The nodes sizes and colors indicate the citation counts and the nodes with larger citation counts are larger and darker.

Comparison

Configurations				Proposed method		Chen & Lei (2018)	
K	n	λ	β	L_2 loss	L_2 loss+stability	L_2 loss	L_2 loss+stability
3	600	15	0.2	0.73	0.87	0.00	0.00
		20	0.2	0.97	0.99	0.02	0.00
		30	0.2	1.00	1.00	0.43	0.40
		40	0.2	1.00	1.00	0.88	0.98
5	600	15	0.2	0.49	0.58	0.00	0.00
		20	0.2	0.90	0.95	0.00	0.00
		30	0.2	0.99	1.00	0.05	0.01
		40	0.2	0.99	1.00	0.27	0.24
5	1200	15	0.2	0.67	0.76	0.00	0.00
		20	0.2	0.99	0.99	0.00	0.00
		30	0.2	1.00	1.00	0.04	0.00
		40	0.2	1.00	1.00	0.41	0.33
3	600	40	0.1	1.00	1.00	0.99	1.00
		40	0.2	1.00	1.00	0.88	0.98
		40	0.5	0.95	0.97	0.00	0.00
5	600	40	0.1	1.00	1.00	0.79	0.96
		40	0.2	0.99	1.00	0.27	0.24
		40	0.5	0.00	0.00	0.00	0.00
5	1200	40	0.1	1.00	1.00	0.90	0.99
		40	0.2	1.00	1.00	0.41	0.33
		40	0.5	0.00	0.00	0.00	0.00

References

Kehui Chen, Jing Lei (2015) Network Cross-Validation for Determining the Number of Communities in Network Data *arXiv*.

Tianxi Li, Elizaveta Levina, Ji Zhu (2020) Network cross-validation by edge sampling *Biometrika*, 107(2): 257-276.

Yuan Zhang, Elizaveta Levina, Ji Zhu (2017). Estimating network edge probabilities by neighbourhood smoothing. *Biometrika*, 104(4): 771-783.

The End