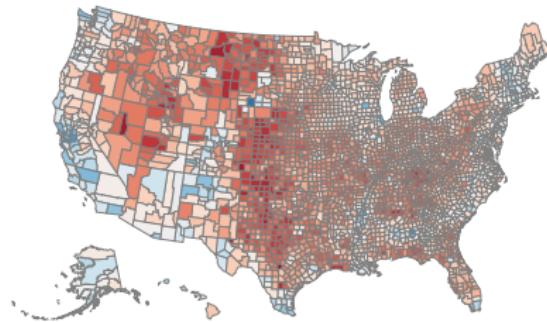


MODELING AND INFERRING ATTRIBUTED GRAPHS

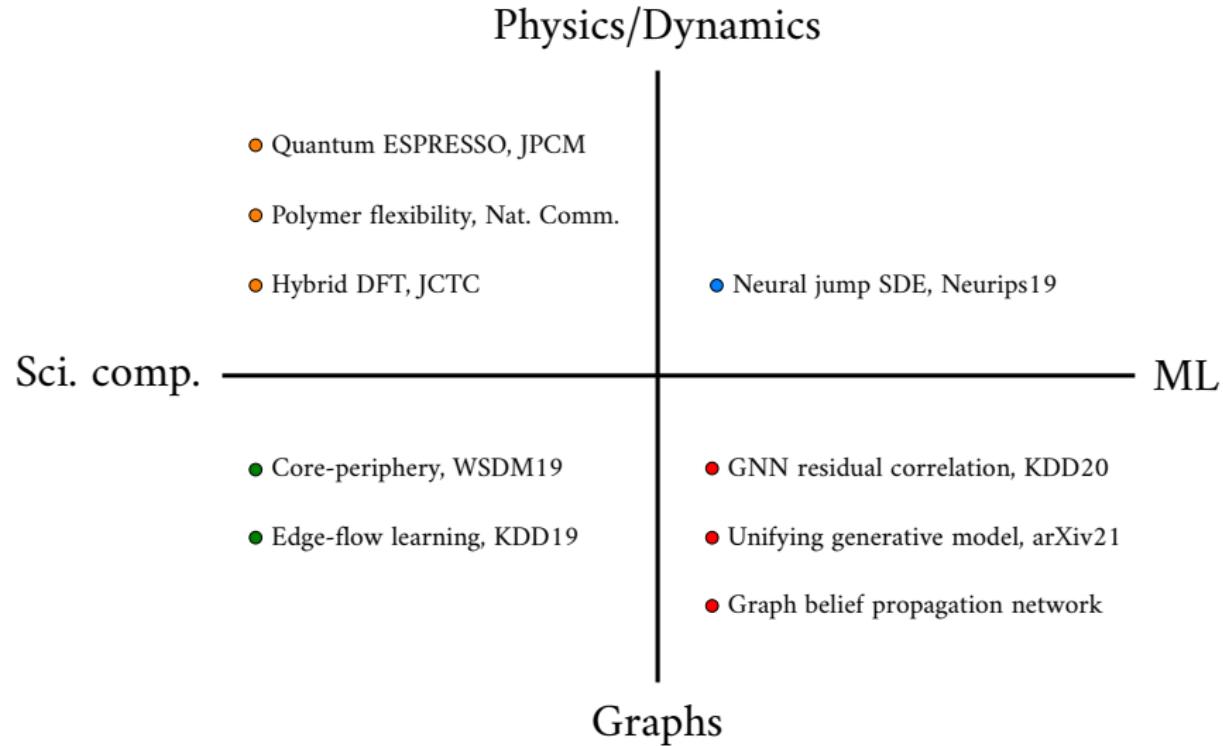


Junteng Jia

Department of Computer Science, Cornell University, Ithaca, NY 14853

Ph.D. THESIS DEFENSE – APRIL 12, 2021

RESEARCH OVERVIEW



RESEARCH OVERVIEW

Physics/Dynamics

Sci. comp.

ML

Graphs



- Quantum ESPRESSO, JPCM
- Polymer flexibility, Nat. Comm.
- Hybrid DFT, JCTC
- Neural jump SDE, Neurips19
- Core-periphery, WSDM19
- Edge-flow learning, KDD19
- GNN residual correlation, KDD20
- Unifying generative model, arXiv21
- Graph belief propagation network

RESEARCH OVERVIEW

Physics/Dynamics

Sci. comp.

ML

Graphs



- Quantum ESPRESSO, JPCM
- Polymer flexibility, Nat. Comm.
- Hybrid DFT, JCTC
- Neural jump SDE, Neurips19
- Core-periphery, WSDM19
- Edge-flow learning, KDD19
- GNN residual correlation, KDD20
- Unifying generative model, arXiv21
- Graph belief propagation network

GRAPHS DATA ARE INCREASINGLY MORE COMPLEX



Social Networks

nodes: people

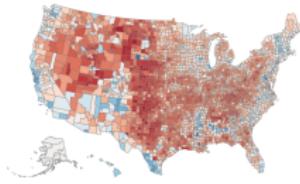
edges: friendships



Finance

nodes: accounts

edges: transactions



Politics

nodes: regions

edges: adjacency



Traffic

nodes: stations

edges: railways

GRAPHS DATA ARE INCREASINGLY MORE COMPLEX



Social Networks

nodes: people

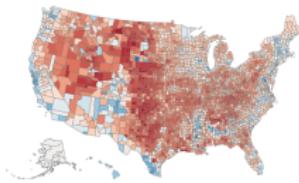
edges: friendships



Finance

nodes: accounts

edges: transactions



Politics

nodes: regions

edges: adjacency



Traffic

nodes: stations

edges: railways

- node attributes contains rich information,

GRAPHS DATA ARE INCREASINGLY MORE COMPLEX



Social Networks

nodes: people

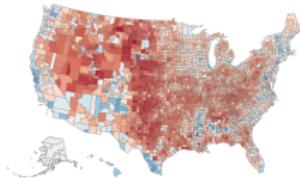
edges: friendships



Finance

nodes: accounts

edges: transactions



Politics

nodes: regions

edges: adjacency



Traffic

nodes: stations

edges: railways

- node attributes contains rich information,
 - social network — user gender, age, post, photo, video, ...
 - politics — local income, unemployment, crime, protest, consumer data, ...
 - finance — opening branch, amount, active time, platform, ...
 - traffic — volume, accidents, number of entrance, population density, ...

GRAPHS DATA ARE INCREASINGLY MORE COMPLEX



Social Networks

nodes: people

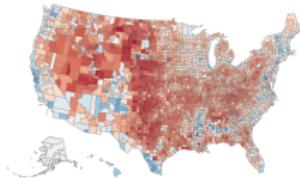
edges: friendships



Finance

nodes: accounts

edges: transactions



Politics

nodes: regions

edges: adjacency



Traffic

nodes: stations

edges: railways

- node attributes contains rich information, useful in real-world activities
 - social network — user gender, age, post, photo, video, ...
 - politics — local income, unemployment, crime, protest, consumer data, ...
 - finance — opening branch, amount, active time, platform, ...
 - traffic — volume, accidents, number of entrance, population density, ...

GRAPHS DATA ARE INCREASINGLY MORE COMPLEX



Social Networks

nodes: people

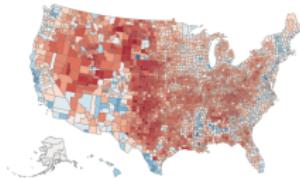
edges: friendships



Finance

nodes: accounts

edges: transactions



Politics

nodes: regions

edges: adjacency



Traffic

nodes: stations

edges: railways

- node attributes contains rich information, useful in real-world activities
 - social network — user gender, age, post, photo, video, ...
 - politics — local income, unemployment, crime, protest, consumer data, ...
 - finance — opening branch, amount, active time, platform, ...
 - traffic — volume, accidents, number of entrance, population density, ...
- node attributes are oftentimes incomplete

WE OFTEN WANT TO PREDICT NODE LABELS



Social Networks

nodes: people

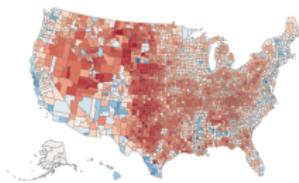
edges: friendships



Finance

nodes: accounts

edges: transactions



Politics

nodes: regions

edges: adjacency



Traffic

nodes: stations

edges: railways

WE OFTEN WANT TO PREDICT NODE LABELS



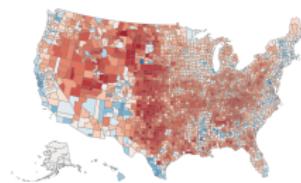
Social Networks

nodes: people
edges: friendships



Finance

nodes: accounts
edges: transactions



Politics

nodes: regions
edges: adjacency



Traffic

nodes: stations
edges: railways

- predicting missing node attributes has many real-world applications

WE OFTEN WANT TO PREDICT NODE LABELS



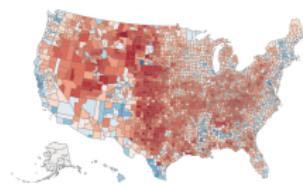
Social Networks

nodes: people
edges: friendships



Finance

nodes: accounts
edges: transactions



Politics

nodes: regions
edges: adjacency



Traffic

nodes: stations
edges: railways

- predicting missing node attributes has many real-world applications
 - gender prediction in social network [Peel 17; Altenburger-Ugander 18]
 - election forecasting in politics [Li+ 19; Jia-Benson 20]
 - fraud detection in finance [Weber+ 18,19; Pareja+ 19]
 - flow prediction in traffic [Zhao+ 18; Jia-Benson 20]

WE OFTEN WANT TO PREDICT NODE LABELS



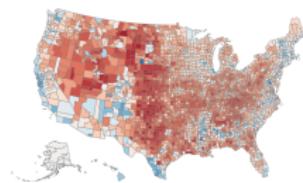
Social Networks

nodes: people
edges: friendships



Finance

nodes: accounts
edges: transactions



Politics

nodes: regions
edges: adjacency



Traffic

nodes: stations
edges: railways

- predicting missing node attributes has many real-world applications
 - gender prediction in social network [Peel 17; Altenburger-Ugander 18]
 - election forecasting in politics [Li+ 19; Jia-Benson 20]
 - fraud detection in finance [Weber+ 18,19; Pareja+ 19]
 - flow prediction in traffic [Zhao+ 18; Jia-Benson 20]
- however, research often driven by improving on benchmark datasets
 - classification in citation/coauthor network [Lu-Getoor 03; Kipf-Welling 17; Shchur+ 18]

REGRESSION VS CLASSIFICATION

- First, we develop **theory** (statistical models) on **regression** problems
Outcomes are real-valued like age, income, temperature ...

REGRESSION VS CLASSIFICATION

- First, we develop **theory** (statistical models) on **regression** problems
Outcomes are real-valued like age, income, temperature ...
- Later, we extend to the **classification** setting
Labels are discrete-valued like cat/dog, male/female ...

NODE LABEL PREDICTION IN ATTRIBUTED GRAPHS

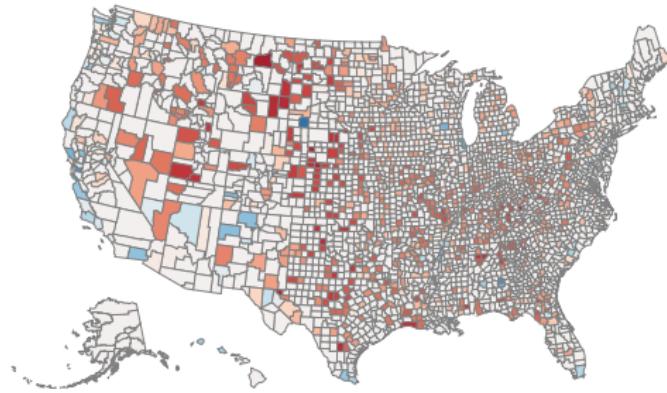
Motivating example — Predicting election from pollsters



- Each county as a node; bordering counties are connected

NODE LABEL PREDICTION IN ATTRIBUTED GRAPHS

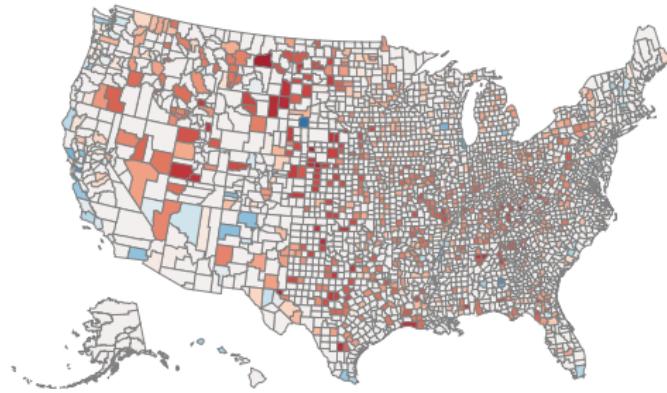
Motivating example — Predicting election from pollsters



- Each county as a node; bordering counties are connected
 - polls: percentage supporters for DEM or GOP, for some counties

NODE LABEL PREDICTION IN ATTRIBUTED GRAPHS

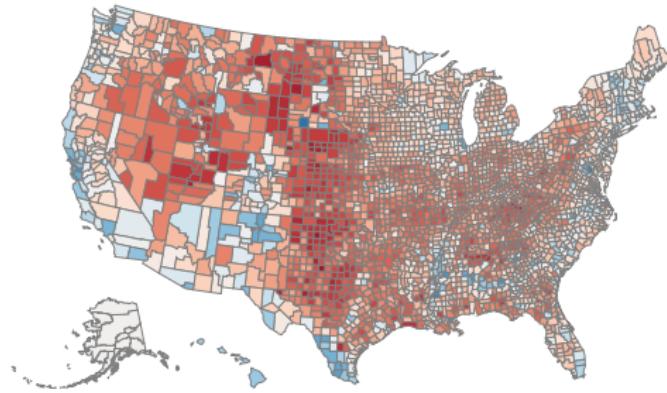
Motivating example — Predicting election from pollsters



- Each county as a node; bordering counties are connected
 - polls: percentage supporters for DEM or GOP, for some counties
 - features: birth/death rate, education, unemployment, income, etc

NODE LABEL PREDICTION IN ATTRIBUTED GRAPHS

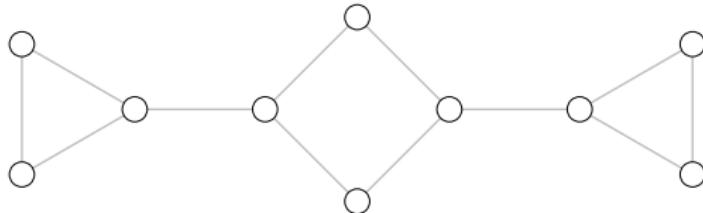
Motivating example — Predicting election from pollsters



- Each county as a node; bordering counties are connected
 - polls: percentage supporters for DEM or GOP, for some counties
 - features: birth/death rate, education, unemployment, income, etc
 - estimate political leaning for other counties

NODE LABEL PREDICTION IN ATTRIBUTED GRAPHS

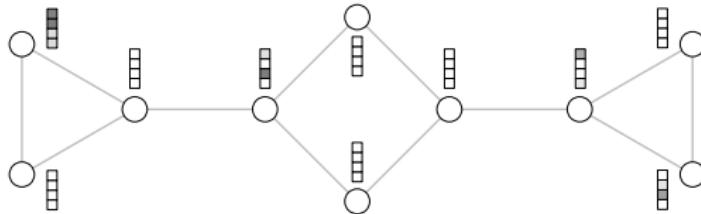
Problem statement



- Problem input:
 - graph topology $G(V, E)$

NODE LABEL PREDICTION IN ATTRIBUTED GRAPHS

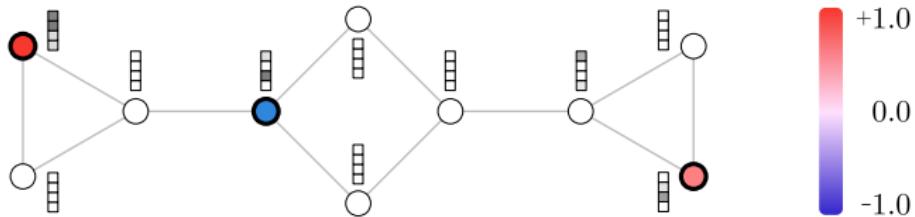
Problem statement



- Problem input:
 - graph topology $G(V, E)$
 - features on all vertices, $\mathbf{X} = [\mathbf{x}_u]_{u \in V}$

NODE LABEL PREDICTION IN ATTRIBUTED GRAPHS

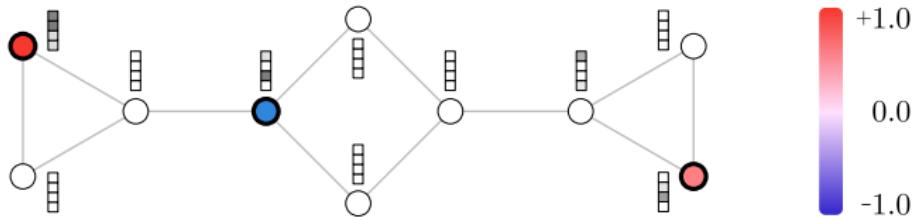
Problem statement



- Problem input:
 - graph topology $G(V, E)$
 - features on all vertices, $\mathbf{X} = [\mathbf{x}_u]_{u \in V}$
 - labels on a subset of vertices $L \subset V$, training labels $\mathbf{y}_L = [y_u]_{u \in L}$

NODE LABEL PREDICTION IN ATTRIBUTED GRAPHS

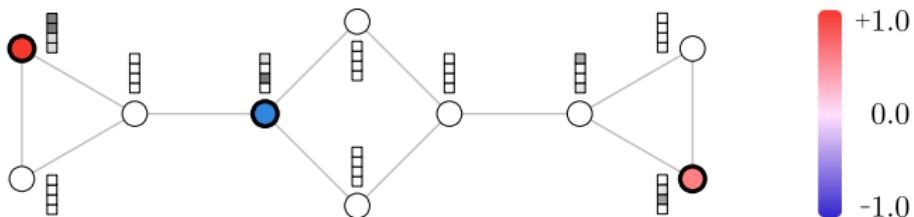
Problem statement



- Problem input:
 - graph topology $G(V, E)$
 - features on all vertices, $\mathbf{X} = [\mathbf{x}_u]_{u \in V}$
 - labels on a subset of vertices $L \subset V$, training labels $\mathbf{y}_L = [y_u]_{u \in L}$
- Problem output:
 - labels on the rest of vertices $U \equiv V \setminus L$, testing labels $\mathbf{y}_U = [y_u]_{u \in U}$

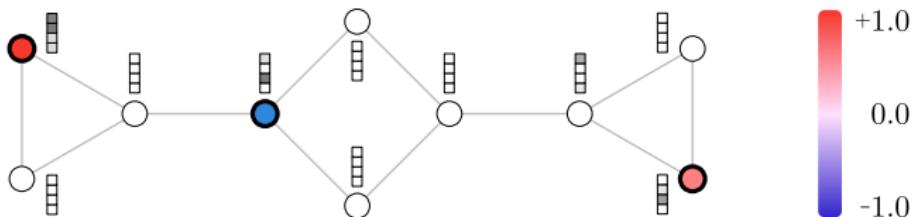
THERE ARE TWO BROAD CLASSES OF METHODS

Label propagation is a transductive method [early 2000s]



THERE ARE TWO BROAD CLASSES OF METHODS

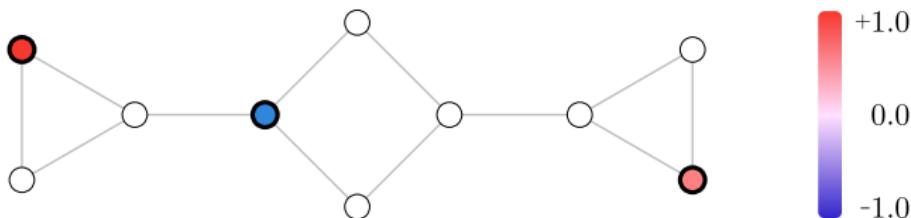
Label propagation is a transductive method [early 2000s]



- assumption: connected nodes have similar labels

THERE ARE TWO BROAD CLASSES OF METHODS

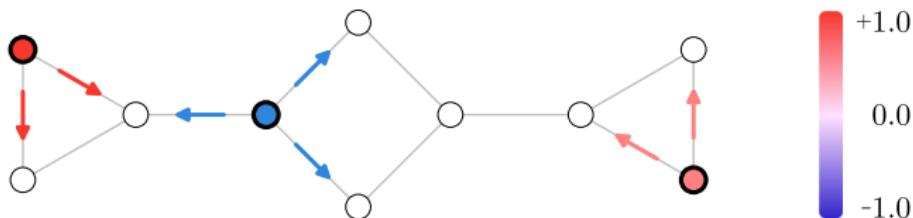
Label propagation is a transductive method [early 2000s]



- assumption: connected nodes have similar labels
 - ignore vertex features,
 - set initial prediction \hat{y} : $\hat{y}_u = y_u$ if observed, otherwise $\hat{y}_u = 0$

THERE ARE TWO BROAD CLASSES OF METHODS

Label propagation is a transductive method [early 2000s]

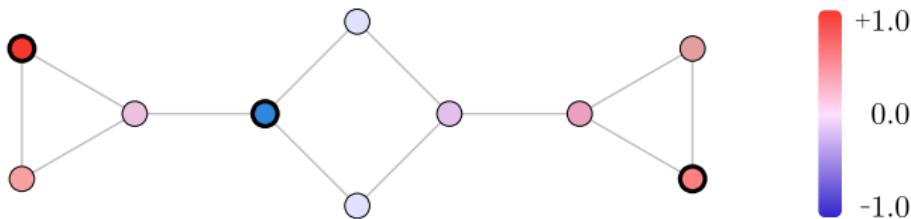


- assumption: connected nodes have similar labels
 - ignore vertex features, diffuse labels along the edges
 - set initial prediction \hat{y} : $\hat{y}_u = y_u$ if observed, otherwise $\hat{y}_u = 0$
 - update: $\mathbf{y}^{(k)} = \alpha \cdot \mathbf{S} \mathbf{y}^{(k-1)} + (1 - \alpha) \cdot \hat{\mathbf{y}}$
 - $\mathbf{S} = \mathbf{D}^{-1/2} \mathbf{A} \mathbf{D}^{-1/2}$ normalized adjacency
 - $0 \leq \alpha \leq 1$ mixing parameter

[Zhu+ 03; Zhou+ 04; Wang-Zhang 06]

THERE ARE TWO BROAD CLASSES OF METHODS

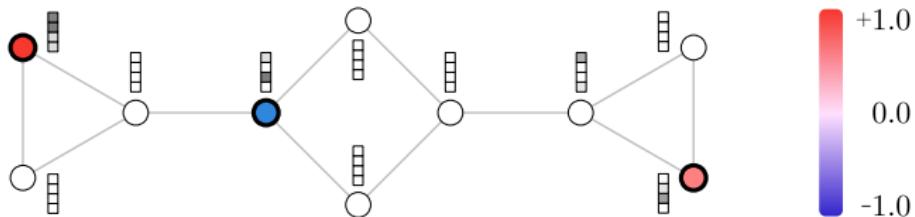
Label propagation is a transductive method [early 2000s]



- assumption: connected nodes have similar labels
 - ignore vertex features, diffuse labels along the edges
 - set initial prediction $\hat{\mathbf{y}}$: $\hat{y}_u = y_u$ if observed, otherwise $\hat{y}_u = 0$
 - update: $\mathbf{y}^{(k)} = \alpha \cdot \mathbf{S}\mathbf{y}^{(k-1)} + (1 - \alpha) \cdot \hat{\mathbf{y}}$
 - $\mathbf{S} = \mathbf{D}^{-1/2} \mathbf{A} \mathbf{D}^{-1/2}$ normalized adjacency
 - $0 \leq \alpha \leq 1$ mixing parameter

THERE ARE TWO BROAD CLASSES OF METHODS

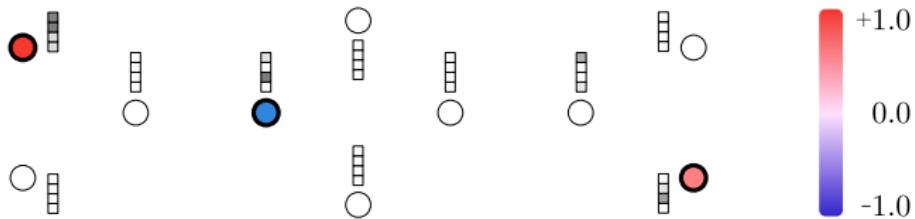
Inductive methods start with naive i.i.d. predictors



- assumption: a label only depend on local features

THERE ARE TWO BROAD CLASSES OF METHODS

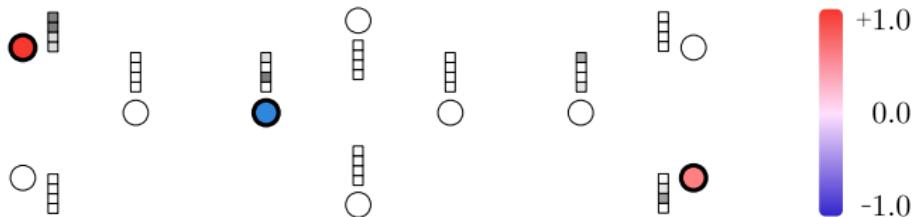
Inductive methods start with naive i.i.d. predictors



- assumption: a label only depend on local features
 - ignore graph structure, treat data points as i.i.d. samples

THERE ARE TWO BROAD CLASSES OF METHODS

Inductive methods start with naive i.i.d. predictors



- assumption: a label only depend on local features
 - ignore graph structure, treat data points as i.i.d. samples
 - e.g. linear regression, multilayer perceptron

THERE ARE TWO BROAD CLASSES OF METHODS

Inductive methods start with naive i.i.d. predictors



- assumption: a label only depend on local features
 - ignore graph structure, treat data points as i.i.d. samples
 - e.g. linear regression, multilayer perceptron
 - training: learn a mapping $f: \mathcal{X} \rightarrow \mathcal{Y}$;

THERE ARE TWO BROAD CLASSES OF METHODS

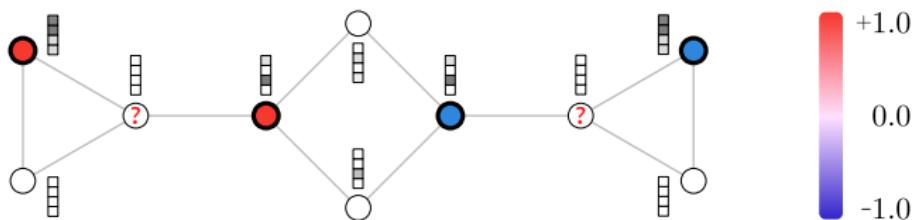
Inductive methods start with naive i.i.d. predictors



- assumption: a label only depend on local features
 - ignore graph structure, treat data points as i.i.d. samples
 - e.g. linear regression, multilayer perceptron
 - training: learn a mapping $f: \mathcal{X} \rightarrow \mathcal{Y}$; inference: $\hat{y}_u = f(\mathbf{x}_u)$

THERE ARE TWO BROAD CLASSES OF METHODS

Inductive methods start with naive i.i.d. predictors



- assumption: a label only depend on local features
 - ignore graph structure, treat data points as i.i.d. samples
 - e.g. linear regression, multilayer perceptron
 - training: learn a mapping $f: \mathcal{X} \rightarrow \mathcal{Y}$; inference: $\hat{y}_u = f(\mathbf{x}_u)$
- problem: correlations between neighbors NOT considered

THERE ARE TWO BROAD CLASSES OF METHODS

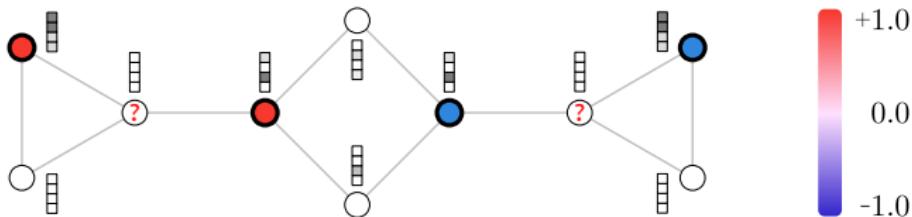
Inductive methods start with naive i.i.d. predictors



- assumption: a label only depend on local features
 - ignore graph structure, treat data points as i.i.d. samples
 - e.g. linear regression, multilayer perceptron
 - training: learn a mapping $f: \mathcal{X} \rightarrow \mathcal{Y}$; inference: $\hat{y}_u = f(\mathbf{x}_u)$
- problem: correlations between neighbors NOT considered
 - vertices with the same features are indistinguishable

THERE ARE TWO BROAD CLASSES OF METHODS

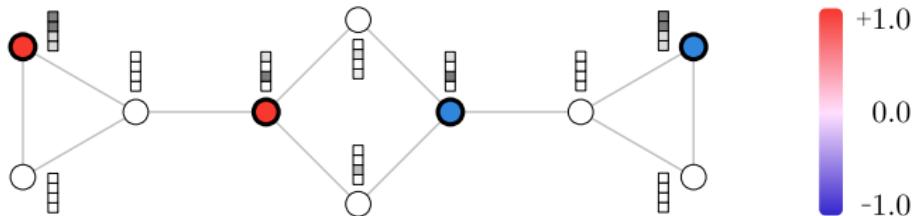
Inductive methods start with naive i.i.d. predictors



- assumption: a label only depend on local features
 - ignore graph structure, treat data points as i.i.d. samples
 - e.g. linear regression, multilayer perceptron
 - training: learn a mapping $f: \mathcal{X} \rightarrow \mathcal{Y}$; inference: $\hat{y}_u = f(\mathbf{x}_u)$
- problem: correlations between neighbors NOT considered
 - vertices with the same features are indistinguishable

THERE ARE TWO BROAD CLASSES OF METHODS

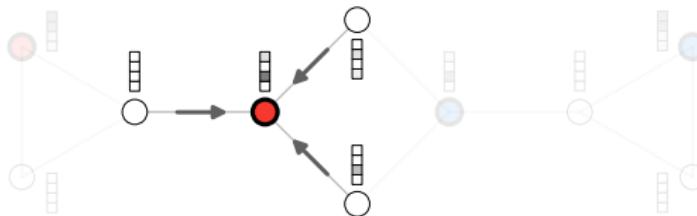
A more sophisticated inductive method is Graph Neural Networks [late 2010s]



- assumption: a label only depend on features in the neighborhood

THERE ARE TWO BROAD CLASSES OF METHODS

A more sophisticated inductive method is Graph Neural Networks [late 2010s]

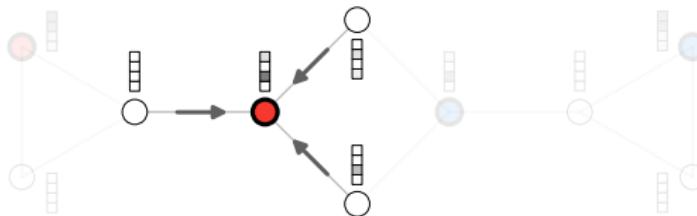


- assumption: a label only depend on features in the neighborhood

$$\mathbf{h}_u^{(0)} = \mathbf{x}_u; \quad \mathbf{h}_u^{(k)} = q \left(\mathbf{h}_u^{(k-1)}, \{\mathbf{h}_v^{(k-1)}\}_{v \in N(u)} \right); \quad \hat{y}_u = g(\mathbf{h}_u^{(K)})$$

THERE ARE TWO BROAD CLASSES OF METHODS

A more sophisticated inductive method is Graph Neural Networks [late 2010s]



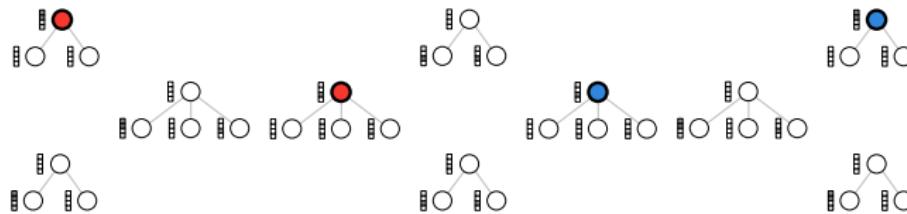
- assumption: a label only depend on features in the neighborhood

$$\mathbf{h}_u^{(0)} = \mathbf{x}_u; \quad \mathbf{h}_u^{(k)} = q \left(\mathbf{h}_u^{(k-1)}, \{\mathbf{h}_v^{(k-1)}\}_{v \in N(u)} \right); \quad \hat{y}_u = g(\mathbf{h}_u^{(K)})$$

- prediction decided by subtree rooted at u

THERE ARE TWO BROAD CLASSES OF METHODS

A more sophisticated inductive method is Graph Neural Networks [late 2010s]



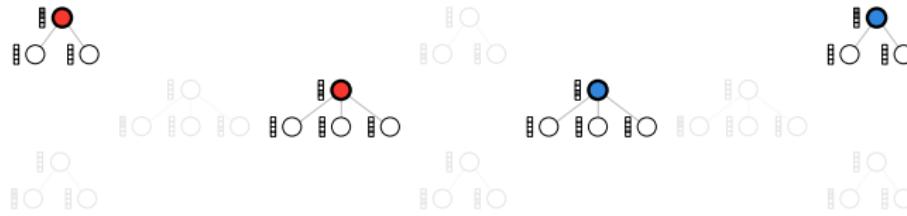
- assumption: a label only depend on features in the neighborhood

$$\mathbf{h}_u^{(0)} = \mathbf{x}_u; \quad \mathbf{h}_u^{(k)} = q\left(\mathbf{h}_u^{(k-1)}, \{\mathbf{h}_v^{(k-1)}\}_{v \in N(u)}\right); \quad \hat{y}_u = g(\mathbf{h}_u^{(K)})$$

- prediction decided by subtree rooted at u

THERE ARE TWO BROAD CLASSES OF METHODS

A more sophisticated inductive method is Graph Neural Networks [late 2010s]



- assumption: a label only depend on features in the neighborhood

$$\mathbf{h}_u^{(0)} = \mathbf{x}_u; \quad \mathbf{h}_u^{(k)} = q \left(\mathbf{h}_u^{(k-1)}, \{\mathbf{h}_v^{(k-1)}\}_{v \in N(u)} \right); \quad \hat{y}_u = g(\mathbf{h}_u^{(K)})$$

- prediction decided by subtree rooted at u
- learn a function $\hat{y}_u = f_{\theta} (\{\mathbf{x}_v\}_{v \in N_K(u)})$;

THERE ARE TWO BROAD CLASSES OF METHODS

A more sophisticated inductive method is Graph Neural Networks [late 2010s]



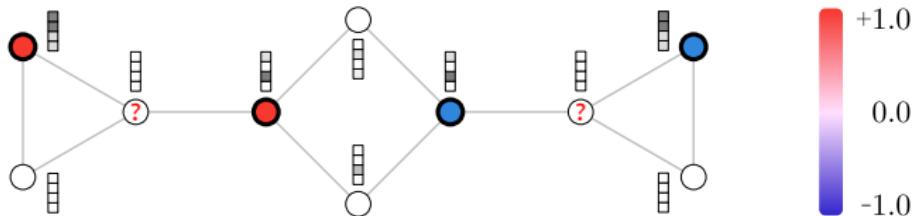
- assumption: a label only depend on features in the neighborhood

$$\mathbf{h}_u^{(0)} = \mathbf{x}_u; \quad \mathbf{h}_u^{(k)} = q \left(\mathbf{h}_u^{(k-1)}, \{\mathbf{h}_v^{(k-1)}\}_{v \in N(u)} \right); \quad \hat{y}_u = g(\mathbf{h}_u^{(K)})$$

- prediction decided by subtree rooted at u
- learn a function $\hat{y}_u = f_{\theta} (\{\mathbf{x}_v\}_{v \in N_K(u)})$; inductive inference

THERE ARE TWO BROAD CLASSES OF METHODS

A more sophisticated inductive method is Graph Neural Networks [late 2010s]



- assumption: a label only depend on features in the neighborhood

$$h_u^{(0)} = \mathbf{x}_u; \quad h_u^{(k)} = q \left(h_u^{(k-1)}, \{h_v^{(k-1)}\}_{v \in N(u)} \right); \quad \hat{y}_u = g(h_u^{(K)})$$

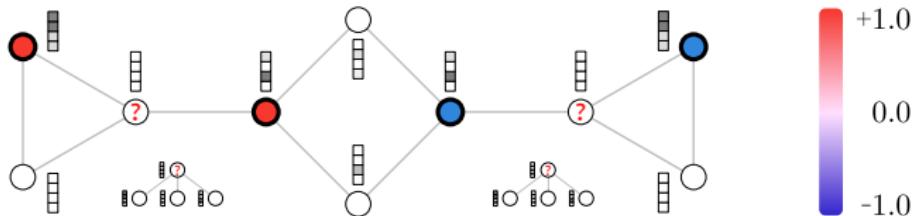
- prediction decided by subtree rooted at u
- learn a function $\hat{y}_u = f_\theta (\{\mathbf{x}_v\}_{v \in N_K(u)})$; inductive inference

- problem: correlations between **neighboring labels** NOT considered

[Kipf-Welling 16; Hamilton+ 17; Zhou+ 18; Velickovic+ 18; Xu+ 19; ~10,000 papers in 5 years]

THERE ARE TWO BROAD CLASSES OF METHODS

A more sophisticated inductive method is Graph Neural Networks [late 2010s]



- assumption: a label only depend on features in the neighborhood

$$\mathbf{h}_u^{(0)} = \mathbf{x}_u; \quad \mathbf{h}_u^{(k)} = q \left(\mathbf{h}_u^{(k-1)}, \{\mathbf{h}_v^{(k-1)}\}_{v \in N(u)} \right); \quad \hat{y}_u = g(\mathbf{h}_u^{(K)})$$

- prediction decided by subtree rooted at u
- learn a function $\hat{y}_u = f_{\theta} (\{\mathbf{x}_v\}_{v \in N_K(u)})$; inductive inference
- problem: correlations between **neighboring labels** NOT considered
 - vertices with the same subtrees are indistinguishable

[Kipf-Welling 16; Hamilton+ 17; Zhou+ 18; Velickovic+ 18; Xu+ 19; ~10,000 papers in 5 years]

COMBINING GNN WITH LABEL PROPAGATION

	ego features	neighboring features	neighboring labels
Label Propagation			●
OLS, MLP	●		
GNN	●	●	

COMBINING GNN WITH LABEL PROPAGATION

	ego features	neighboring features	neighboring labels
Label Propagation			●
OLS, MLP	●		
GNN	●	●	
???	●	●	●

COMBING GNN WITH LABEL PROPAGATION

Can we combine an inductive GNN
with Label Propagation?

COMBING GNN WITH LABEL PROPAGATION

Can we combine an inductive GNN
with Label Propagation?

- decompose outcomes as **learnable from features** and **correlated residuals**

$$y_u = f(\{\mathbf{x}_v\}_{v \in N_K(u)}) + r_u$$

COMBING GNN WITH LABEL PROPAGATION

Can we combine an inductive GNN
with Label Propagation?

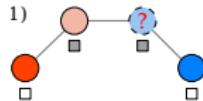
- decompose outcomes as **learnable from features** and **correlated residuals**

$$y_u = f(\{\mathbf{x}_v\}_{v \in N_K(u)}) + r_u$$

conceptually similar to boosting; the two “weak predictors” uses different inputs

COMBING GNN WITH LABEL PROPAGATION

Ground Truth Labels $\{y_u\}_{u \in V}$



- decompose outcomes as **learnable from features** and **correlated residuals**

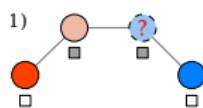
$$y_u = f(\{\mathbf{x}_v\}_{v \in N_K(u)}) + r_u$$

conceptually similar to boosting; the two “weak predictors” uses different inputs

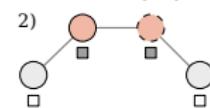
- sequentially deploy **GNN** and **label propagation**

COMBINING GNN WITH LABEL PROPAGATION

Ground Truth Labels $\{y_u\}_{u \in V}$



Predictions $\{\hat{y}_u\}_{u \in V}$



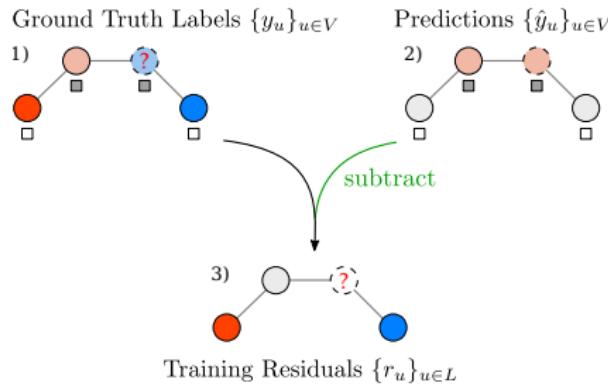
- decompose outcomes as **learnable from features** and **correlated residuals**

$$y_u = f(\{\mathbf{x}_v\}_{v \in N_K(u)}) + r_u$$

conceptually similar to boosting; the two “weak predictors” uses different inputs

- sequentially deploy **GNN** and **label propagation**
 - train a GNN inductively;

COMBINING GNN WITH LABEL PROPAGATION



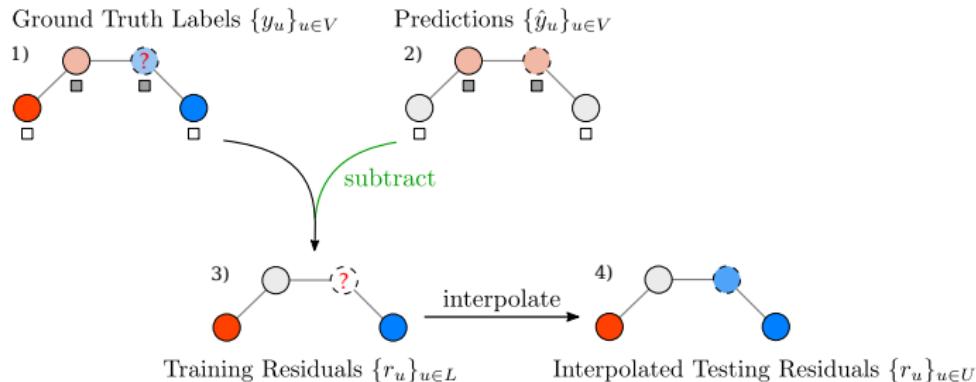
- decompose outcomes as **learnable from features** and **correlated residuals**

$$y_u = f(\{\mathbf{x}_v\}_{v \in N_K(u)}) + r_u$$

conceptually similar to boosting; the two “weak predictors” uses different inputs

- sequentially deploy **GNN** and **label propagation**
 - train a GNN inductively; compute training residual;

COMBINING GNN WITH LABEL PROPAGATION



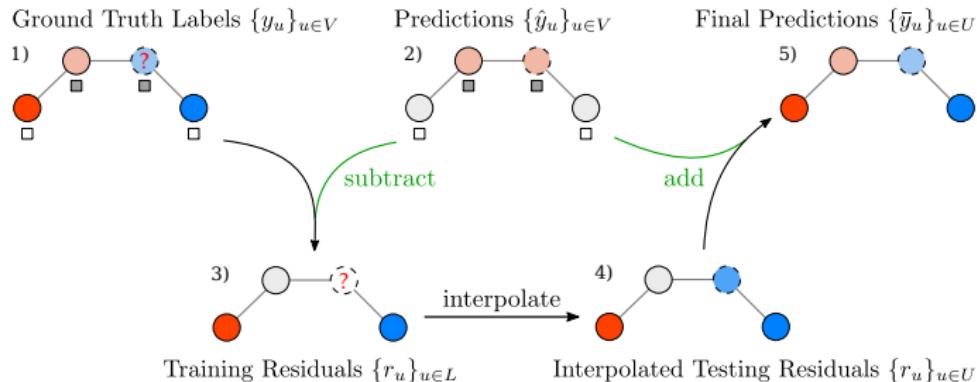
- decompose outcomes as **learnable from features** and **correlated residuals**

$$y_u = f(\{\mathbf{x}_v\}_{v \in N_K(u)}) + r_u$$

conceptually similar to boosting; the two “weak predictors” uses different inputs

- sequentially deploy **GNN** and **label propagation**
 - train a GNN inductively; compute training residual; estimate testing residual

COMBINING GNN WITH LABEL PROPAGATION



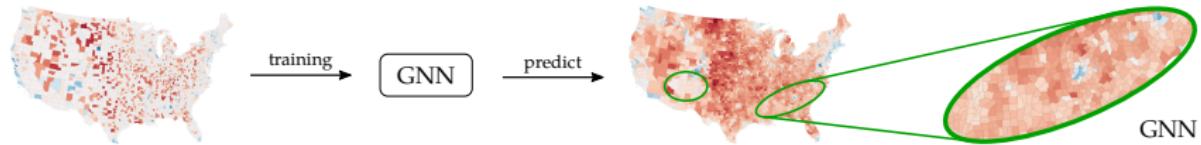
- decompose outcomes as learnable from features and correlated residuals

$$y_u = f(\{\mathbf{x}_v\}_{v \in N_K(u)}) + r_u$$

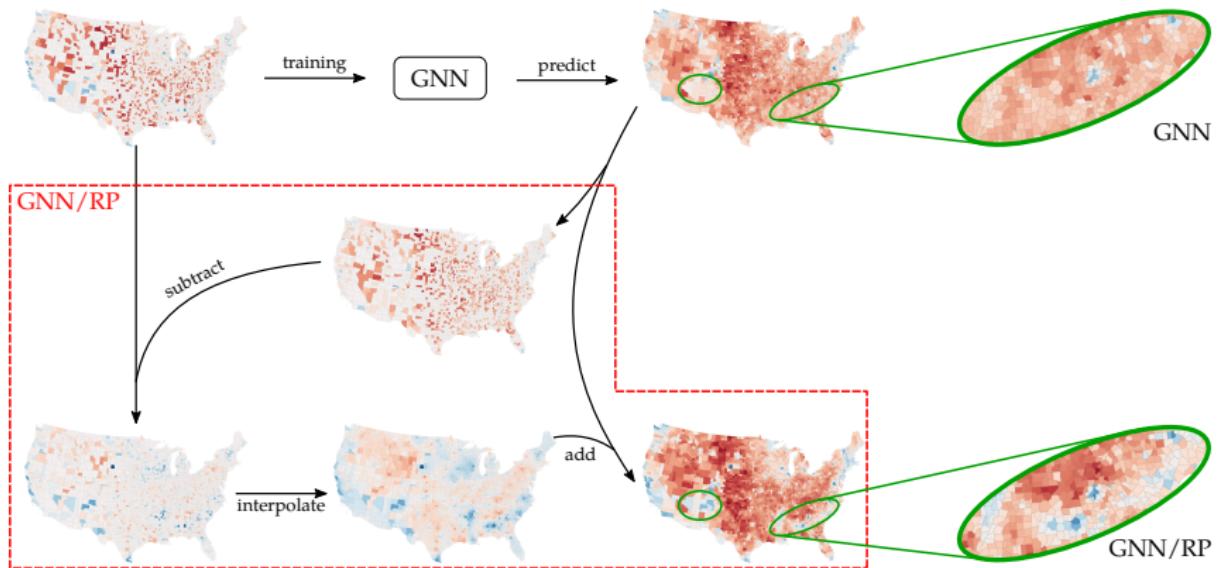
conceptually similar to boosting; the two “weak predictors” uses different inputs

- sequentially deploy GNN and label propagation
 - train a GNN inductively; compute training residual; estimate testing residual
 - final prediction = inductive prediction + estimated residual

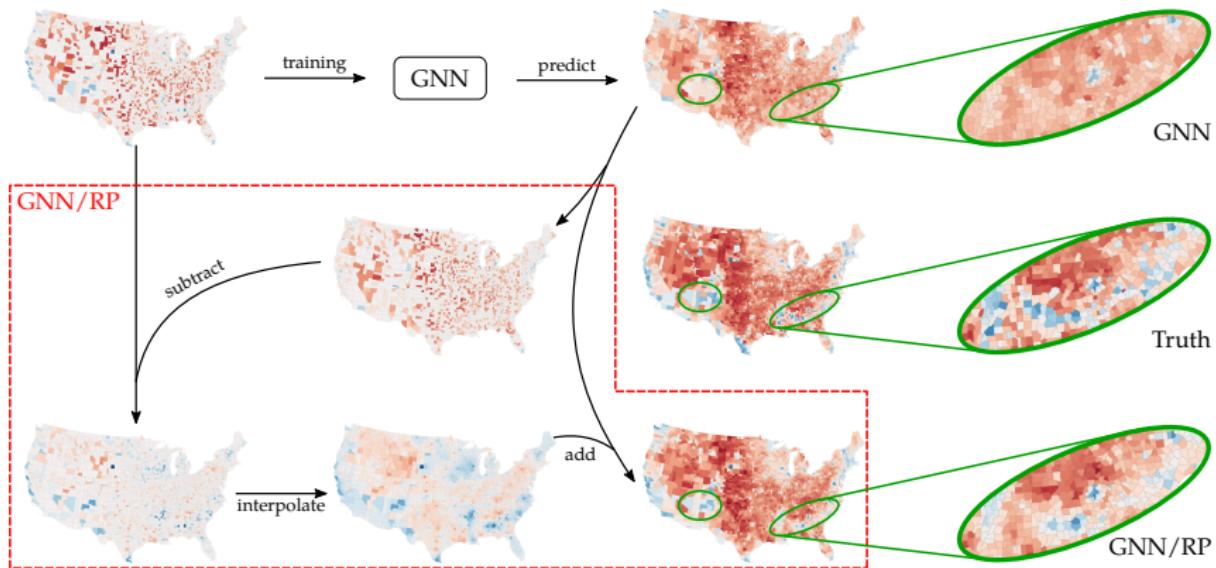
GNN/RP WORKS SUPER WELL IN PRACTICE



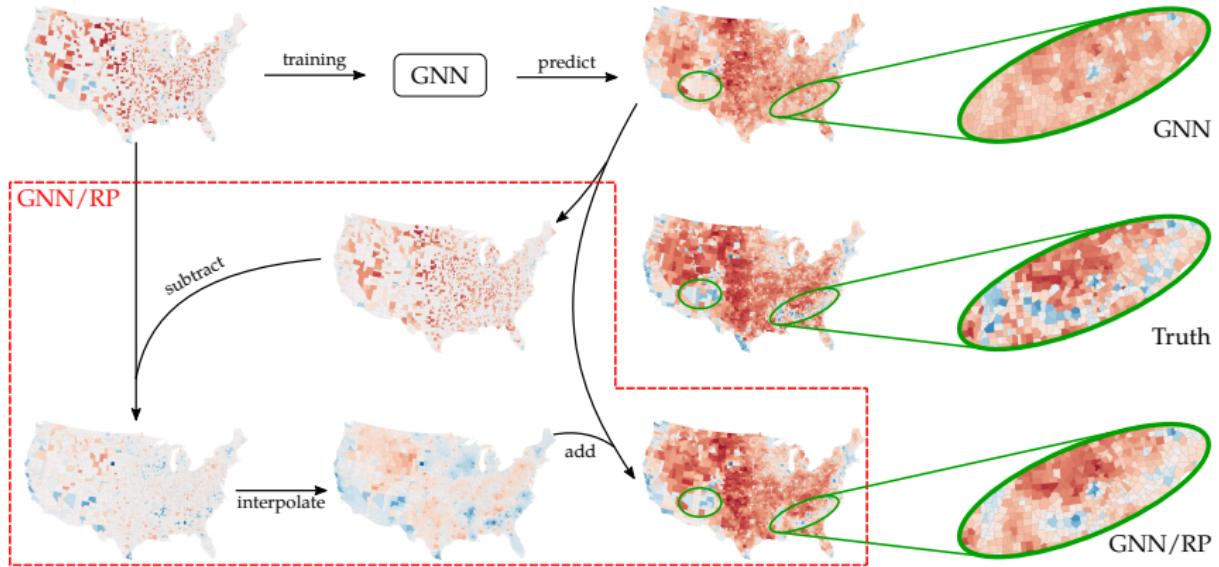
GNN/RP WORKS SUPER WELL IN PRACTICE



GNN/RP WORKS SUPER WELL IN PRACTICE



GNN/RP WORKS SUPER WELL IN PRACTICE



- coefficient of determination (R^2) increases from 0.51 to 0.69 on test set

RP ASSUMES LINEARLY CORRELATED RESIDUALS

	ego features	neighboring features	neighboring labels
Label Propagation			●
OLS, MLP	●		
GNN	●	●	
GNN/RP	●	●	●

RP ASSUMES LINEARLY CORRELATED RESIDUALS

	ego features	neighboring features	neighboring labels
Label Propagation			●
OLS, MLP	●		
GNN	●	●	
GNN/RP	●	●	●

- for regression, the typically training objective is the sum-of-square loss

$$\min_{\theta} \sum_{u \in L} (y_u - \hat{y}_u)^2, \quad \hat{y}_u = f(\mathbf{x}_u, \{\mathbf{x}_v : v \in N_K(u)\}; \theta)$$

RP ASSUMES LINEARLY CORRELATED RESIDUALS

	ego features	neighboring features	neighboring labels
Label Propagation			●
OLS, MLP	●		
GNN	●	●	
GNN/RP	●	●	●

- for regression, the typically training objective is the sum-of-square loss

$$\min_{\theta} \sum_{u \in L} (y_u - \hat{y}_u)^2, \quad \hat{y}_u = f(\mathbf{x}_u, \{\mathbf{x}_v : v \in N_K(u)\}; \theta)$$

corresponds to MLE assuming i.i.d. errors $\mathbf{y} = \hat{\mathbf{y}} + \mathbf{r}$, $\mathbf{r} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$

RP ASSUMES LINEARLY CORRELATED RESIDUALS

	ego features	neighboring features	neighboring labels
Label Propagation			●
OLS, MLP	●		
GNN	●	●	
GNN/RP	●	●	●

- for regression, the typically training objective is the sum-of-square loss

$$\min_{\theta} \sum_{u \in L} (y_u - \hat{y}_u)^2, \quad \hat{y}_u = f(\mathbf{x}_u, \{\mathbf{x}_v : v \in N_K(u)\}; \theta)$$

corresponds to MLE assuming i.i.d. errors $\mathbf{y} = \hat{\mathbf{y}} + \mathbf{r}$, $\mathbf{r} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$

- residual propagation assumes correlated errors $\mathbf{r} \sim \mathcal{N}(\mathbf{0}, (\mathbf{I} - \alpha \mathbf{S})^{-1})$

RP ASSUMES LINEARLY CORRELATED RESIDUALS

	ego features	neighboring features	neighboring labels
Label Propagation			●
OLS, MLP	●		
GNN	●	●	
GNN/RP	●	●	●

- for regression, the typically training objective is the sum-of-square loss

$$\min_{\theta} \sum_{u \in L} (y_u - \hat{y}_u)^2, \quad \hat{y}_u = f(\mathbf{x}_u, \{\mathbf{x}_v : v \in N_K(u)\}; \theta)$$

corresponds to MLE assuming i.i.d. errors $\mathbf{y} = \hat{\mathbf{y}} + \mathbf{r}$, $\mathbf{r} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$

- residual propagation assumes correlated errors $\mathbf{r} \sim \mathcal{N}(\mathbf{0}, (\mathbf{I} - \alpha \mathbf{S})^{-1})$
 - $\mathbf{S} = \mathbf{D}^{-1/2} \mathbf{W} \mathbf{D}^{-1/2}$ is the normalized adjacency matrix

RP ASSUMES LINEARLY CORRELATED RESIDUALS

	ego features	neighboring features	neighboring labels
Label Propagation			●
OLS, MLP	●		
GNN	●	●	
GNN/RP	●	●	●

- for regression, the typically training objective is the sum-of-square loss

$$\min_{\theta} \sum_{u \in L} (y_u - \hat{y}_u)^2, \quad \hat{y}_u = f(\mathbf{x}_u, \{\mathbf{x}_v : v \in N_K(u)\}; \theta)$$

corresponds to MLE assuming i.i.d. errors $\mathbf{y} = \hat{\mathbf{y}} + \mathbf{r}$, $\mathbf{r} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$

- residual propagation assumes correlated errors $\mathbf{r} \sim \mathcal{N}(\mathbf{0}, (\mathbf{I} - \alpha \mathbf{S})^{-1})$
 - $\mathbf{S} = \mathbf{D}^{-1/2} \mathbf{W} \mathbf{D}^{-1/2}$ is the normalized adjacency matrix
 - residual propagation computes expectation of \mathbf{r}_U conditioned on \mathbf{r}_L

RP ASSUMES LINEARLY CORRELATED RESIDUALS

	ego features	neighboring features	neighboring labels
Label Propagation			●
OLS, MLP	●		
GNN	●	●	
GNN/RP	●	●	●

- for regression, the typically training objective is the sum-of-square loss

$$\min_{\theta} \sum_{u \in L} (y_u - \hat{y}_u)^2, \quad \hat{y}_u = f(\mathbf{x}_u, \{\mathbf{x}_v : v \in N_K(u)\}; \theta)$$

corresponds to MLE assuming i.i.d. errors $\mathbf{y} = \hat{\mathbf{y}} + \mathbf{r}$, $\mathbf{r} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$

- residual propagation assumes correlated errors $\mathbf{r} \sim \mathcal{N}(\mathbf{0}, (\mathbf{I} - \alpha \mathbf{S})^{-1})$
 - $\mathbf{S} = \mathbf{D}^{-1/2} \mathbf{W} \mathbf{D}^{-1/2}$ is the normalized adjacency matrix
 - residual propagation computes expectation of \mathbf{r}_U conditioned on \mathbf{r}_L

RP ASSUMES LINEARLY CORRELATED RESIDUALS

	ego features	neighboring features	neighboring labels
Label Propagation			✓
OLS, MLP	●		
GNN	●	●	
GNN/RP	●	●	✓

- for regression, the typically training objective is the sum-of-square loss

$$\min_{\theta} \sum_{u \in L} (y_u - \hat{y}_u)^2, \quad \hat{y}_u = f(\mathbf{x}_u, \{\mathbf{x}_v : v \in N_K(u)\}; \theta)$$

corresponds to MLE assuming i.i.d. errors $\mathbf{y} = \hat{\mathbf{y}} + \mathbf{r}$, $\mathbf{r} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$

- residual propagation assumes correlated errors $\mathbf{r} \sim \mathcal{N}(\mathbf{0}, (\mathbf{I} - \alpha \mathbf{S})^{-1})$
 - $\mathbf{S} = \mathbf{D}^{-1/2} \mathbf{W} \mathbf{D}^{-1/2}$ is the normalized adjacency matrix
 - residual propagation computes expectation of \mathbf{r}_U conditioned on \mathbf{r}_L

RP ASSUMES LINEARLY CORRELATED RESIDUALS

	ego features	neighboring features	neighboring labels
Label Propagation			✓
OLS, MLP	✓		
GNN	✓	✓	
GNN/RP	✓	✓	✓

- for regression, the typically training objective is the sum-of-square loss

$$\min_{\theta} \sum_{u \in L} (y_u - \hat{y}_u)^2, \quad \hat{y}_u = f(\mathbf{x}_u, \{\mathbf{x}_v : v \in N_K(u)\}; \theta)$$

corresponds to MLE assuming i.i.d. errors $\mathbf{y} = \hat{\mathbf{y}} + \mathbf{r}$, $\mathbf{r} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$

- residual propagation assumes correlated errors $\mathbf{r} \sim \mathcal{N}(\mathbf{0}, (\mathbf{I} - \alpha \mathbf{S})^{-1})$
 - $\mathbf{S} = \mathbf{D}^{-1/2} \mathbf{W} \mathbf{D}^{-1/2}$ is the normalized adjacency matrix
 - residual propagation computes expectation of \mathbf{r}_U conditioned on \mathbf{r}_L

RP ASSUMES LINEARLY CORRELATED RESIDUALS

	ego features	neighboring features	neighboring labels
Label Propagation			✓
OLS, MLP	✓		
GNN	✓	?	
GNN/RP	✓	?	✓

- for regression, the typically training objective is the sum-of-square loss

$$\min_{\theta} \sum_{u \in L} (y_u - \hat{y}_u)^2, \quad \hat{y}_u = f(\mathbf{x}_u, \{\mathbf{x}_v : v \in N_K(u)\}; \theta)$$

corresponds to MLE assuming i.i.d. errors $\mathbf{y} = \hat{\mathbf{y}} + \mathbf{r}$, $\mathbf{r} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$

- residual propagation assumes correlated errors $\mathbf{r} \sim \mathcal{N}(\mathbf{0}, (\mathbf{I} - \alpha \mathbf{S})^{-1})$
 - $\mathbf{S} = \mathbf{D}^{-1/2} \mathbf{W} \mathbf{D}^{-1/2}$ is the normalized adjacency matrix
 - residual propagation computes expectation of \mathbf{r}_U conditioned on \mathbf{r}_L

STATISTICAL MODEL FOR NODE ATTRIBUTES

**Is there a common statistical framework
that formalizes all three correlations?**

STATISTICAL MODEL FOR NODE ATTRIBUTES

- Problem input:
 - graph topology $G(V, E)$
 - features on all vertices, $\mathbf{X} = [\mathbf{x}_u]_{u \in V}$
 - labels on a subset of vertices $L \subset V$, training labels $\mathbf{y}_L = [y_u]_{u \in L}$
- Problem output:
 - labels on the rest of vertices $U \equiv V \setminus L$, testing labels $\mathbf{y}_U = [y_u]_{u \in U}$

STATISTICAL MODEL FOR NODE ATTRIBUTES

- Problem input:
 - graph topology $G(V, E)$
 - features on all vertices, $\mathbf{X} = [\mathbf{x}_u]_{u \in V}$
 - labels on a subset of vertices $L \subset V$, training labels $\mathbf{y}_L = [y_u]_{u \in L}$
- Problem output:
 - labels on the rest of vertices $U \equiv V \setminus L$, testing labels $\mathbf{y}_U = [y_u]_{u \in U}$
- Problem solution:

STATISTICAL MODEL FOR NODE ATTRIBUTES

- Problem input:
 - graph topology $G(V, E)$
 - features on all vertices, $\mathbf{X} = [\mathbf{x}_u]_{u \in V}$
 - labels on a subset of vertices $L \subset V$, training labels $\mathbf{y}_L = [y_u]_{u \in L}$
- Problem output:
 - labels on the rest of vertices $U \equiv V \setminus L$, testing labels $\mathbf{y}_U = [y_u]_{u \in U}$
- Problem solution:
 - specify a generative model (joint distribution of all node attributes)

STATISTICAL MODEL FOR NODE ATTRIBUTES

- Problem input:
 - graph topology $G(V, E)$
 - features on all vertices, $\mathbf{X} = [\mathbf{x}_u]_{u \in V}$
 - labels on a subset of vertices $L \subset V$, training labels $\mathbf{y}_L = [y_u]_{u \in L}$
- Problem output:
 - labels on the rest of vertices $U \equiv V \setminus L$, testing labels $\mathbf{y}_U = [y_u]_{u \in U}$
- Problem solution:
 - specify a generative model (joint distribution of all node attributes)
 - inference with conditional expectation $\mathbf{y}_U = E[\mathbf{y}_U | \text{observations}]$

STATISTICAL MODEL FOR NODE ATTRIBUTES

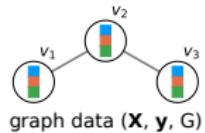
- Problem input:
 - graph topology $G(V, E)$
 - features on all vertices, $\mathbf{X} = [\mathbf{x}_u]_{u \in V}$
 - labels on a subset of vertices $L \subset V$, training labels $\mathbf{y}_L = [y_u]_{u \in L}$
- Problem output:
 - labels on the rest of vertices $U \equiv V \setminus L$, testing labels $\mathbf{y}_U = [y_u]_{u \in U}$
- Problem solution:
 - specify a generative model (joint distribution of all node attributes)
 - inference with conditional expectation $\mathbf{y}_U = E[\mathbf{y}_U | \text{observations}]$
 - different observations gives different algorithms

STATISTICAL MODEL FOR NODE ATTRIBUTES

Data Type

Corresponding Gaussian MRF

Learning Algorithm



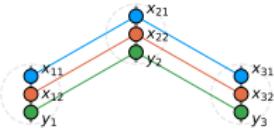
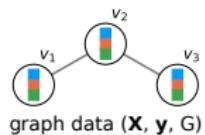
graph data (\mathbf{X} , \mathbf{y} , G)

STATISTICAL MODEL FOR NODE ATTRIBUTES

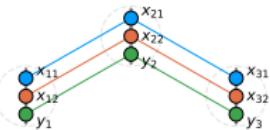
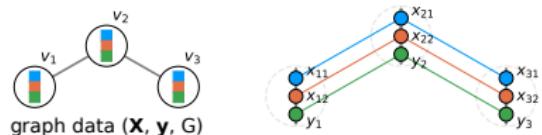
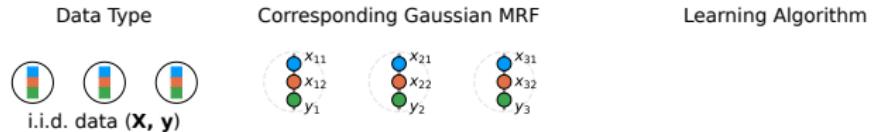
Data Type

Corresponding Gaussian MRF

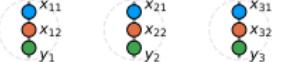
Learning Algorithm

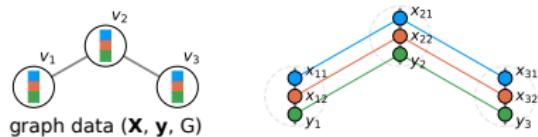


STATISTICAL MODEL FOR NODE ATTRIBUTES



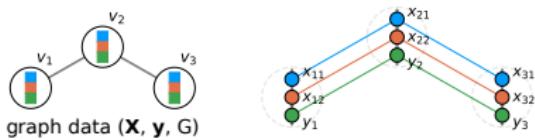
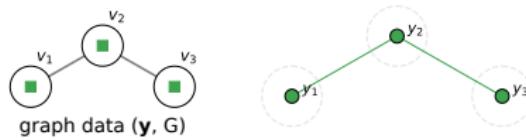
STATISTICAL MODEL FOR NODE ATTRIBUTES

Data Type	Corresponding Gaussian MRF	Learning Algorithm
i.i.d. data (\mathbf{X}, \mathbf{y})		<p>linear regression $E[\mathbf{y}_U \mathbf{X}] = \mathbf{X}_U \boldsymbol{\beta}$ $\boldsymbol{\beta} = (\mathbf{X}_L^\top \mathbf{X}_L)^{-1} \mathbf{X}_L^\top \mathbf{y}_L$</p> <p>condition on \mathbf{X} e.g. $L = \{1, 3\}$, $U = \{2\}$</p>



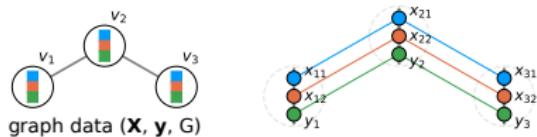
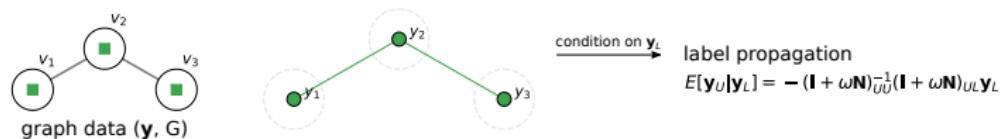
STATISTICAL MODEL FOR NODE ATTRIBUTES

Data Type	Corresponding Gaussian MRF	Learning Algorithm
i.i.d. data (\mathbf{X}, \mathbf{y})		<p>linear regression $E[\mathbf{y}_U \mathbf{X}] = \mathbf{X}_U \boldsymbol{\beta}$ $\boldsymbol{\beta} = (\mathbf{X}_L^\top \mathbf{X}_L)^{-1} \mathbf{X}_L^\top \mathbf{y}_L$</p> <p>condition on \mathbf{X} e.g. $L = \{1, 3\}$, $U = \{2\}$</p>

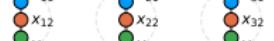
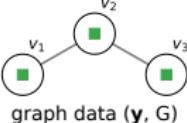
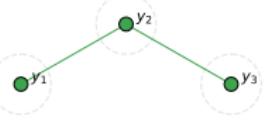
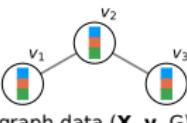
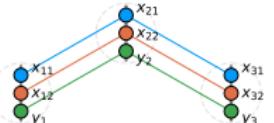


STATISTICAL MODEL FOR NODE ATTRIBUTES

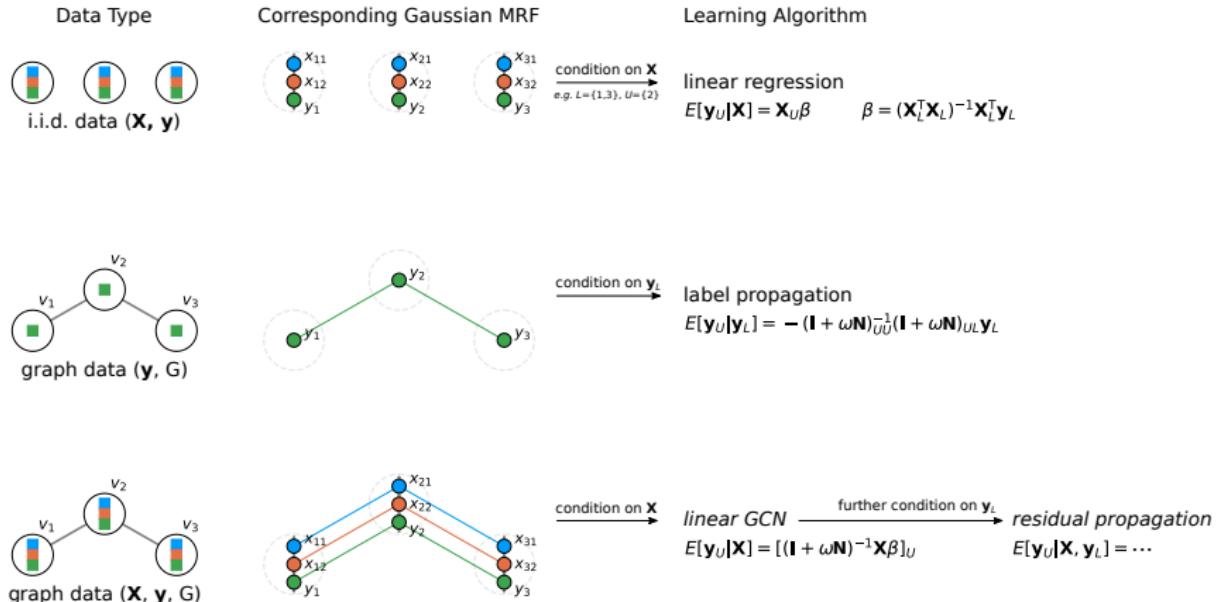
Data Type	Corresponding Gaussian MRF	Learning Algorithm
i.i.d. data (\mathbf{X}, \mathbf{y})		$\xrightarrow{\text{condition on } \mathbf{X} \\ \text{e.g. } L=\{1,3\}, U=\{2\}}$ linear regression $E[\mathbf{y}_U \mathbf{X}] = \mathbf{X}_U \boldsymbol{\beta}$ $\boldsymbol{\beta} = (\mathbf{X}_L^\top \mathbf{X}_L)^{-1} \mathbf{X}_L^\top \mathbf{y}_L$



STATISTICAL MODEL FOR NODE ATTRIBUTES

Data Type	Corresponding Gaussian MRF	Learning Algorithm
 i.i.d. data (\mathbf{X}, \mathbf{y})		$\xrightarrow{\text{condition on } \mathbf{X}}$ e.g. $L = \{1, 3\}$, $U = \{2\}$ linear regression $E[\mathbf{y}_U \mathbf{X}] = \mathbf{X}_U \boldsymbol{\beta}$ $\boldsymbol{\beta} = (\mathbf{X}_L^\top \mathbf{X}_L)^{-1} \mathbf{X}_L^\top \mathbf{y}_L$
 graph data (\mathbf{y}, G)		$\xrightarrow{\text{condition on } \mathbf{y}_L}$ label propagation $E[\mathbf{y}_U \mathbf{y}_L] = -(\mathbf{I} + \omega \mathbf{N})_{UU}^{-1} (\mathbf{I} + \omega \mathbf{N})_{UL} \mathbf{y}_L$
 graph data $(\mathbf{X}, \mathbf{y}, G)$		$\xrightarrow{\text{condition on } \mathbf{X}}$ linear GCN $E[\mathbf{y}_U \mathbf{X}] = [(\mathbf{I} + \omega \mathbf{N})^{-1} \mathbf{X} \boldsymbol{\beta}]_U$

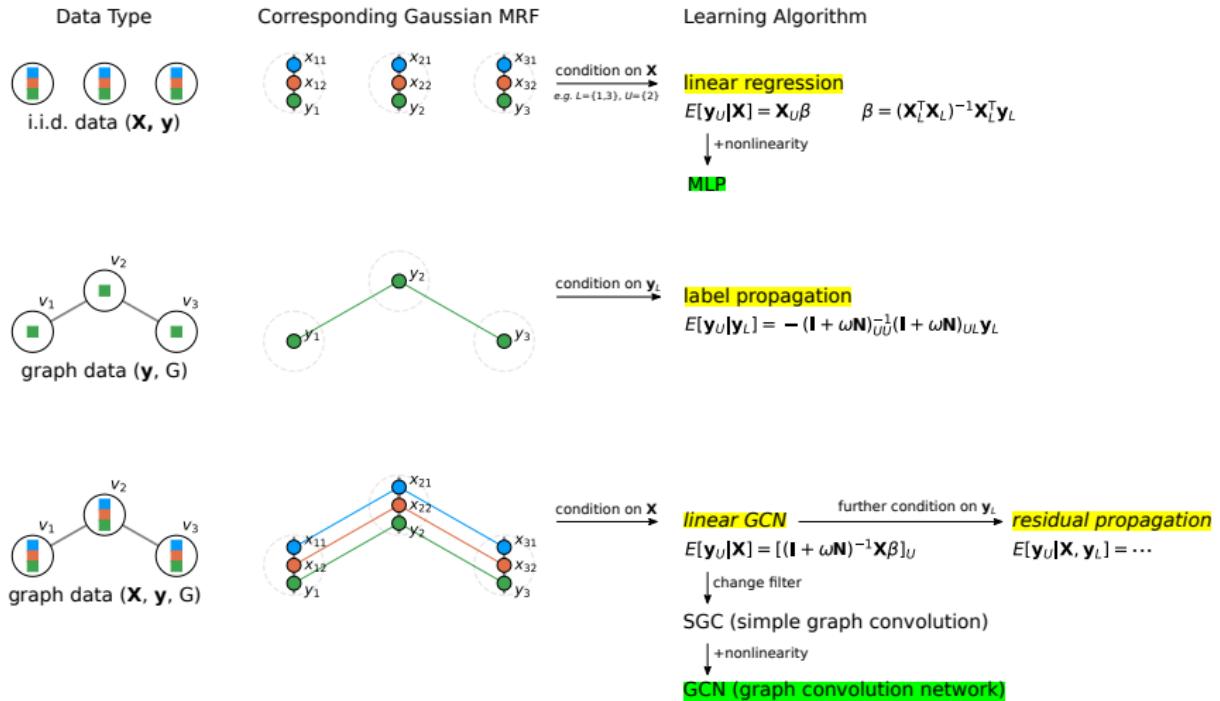
STATISTICAL MODEL FOR NODE ATTRIBUTES



STATISTICAL MODEL FOR NODE ATTRIBUTES

Data Type	Corresponding Gaussian MRF	Learning Algorithm
i.i.d. data (\mathbf{X}, \mathbf{y})		$\xrightarrow{\text{condition on } \mathbf{X}}$ e.g. $L = \{1, 3\}$, $U = \{2\}$ linear regression $E[\mathbf{y}_U \mathbf{X}] = \mathbf{X}_U \boldsymbol{\beta}$ $\boldsymbol{\beta} = (\mathbf{X}_L^\top \mathbf{X}_L)^{-1} \mathbf{X}_L^\top \mathbf{y}_L$
graph data (\mathbf{y}, G)		$\xrightarrow{\text{condition on } \mathbf{y}_L}$ label propagation $E[\mathbf{y}_U \mathbf{y}_L] = -(\mathbf{I} + \omega \mathbf{N})_{UU}^{-1} (\mathbf{I} + \omega \mathbf{N})_{UL} \mathbf{y}_L$
graph data $(\mathbf{X}, \mathbf{y}, G)$		$\xrightarrow{\text{condition on } \mathbf{X}}$ linear GCN $\xrightarrow{\text{further condition on } \mathbf{y}_L}$ residual propagation $E[\mathbf{y}_U \mathbf{X}] = [(\mathbf{I} + \omega \mathbf{N})^{-1} \mathbf{X} \boldsymbol{\beta}]_U$ $E[\mathbf{y}_U \mathbf{X}, \mathbf{y}_L] = \dots$

STATISTICAL MODEL FOR NODE ATTRIBUTES



STATISTICAL MODEL FOR NODE ATTRIBUTES

The joint distribution of all node attributes is a Gaussian MRF

- Real-valued attributes $\mathbf{a}_u = [\mathbf{x}_u; y_u] \in \mathbb{R}^{p+1}$ on each node u .
- $A_i = i$ th attribute over all nodes.
- $\mathbf{N} = \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{W}\mathbf{D}^{-1/2}$ is the normalized Laplacian.

STATISTICAL MODEL FOR NODE ATTRIBUTES

The joint distribution of all node attributes is a Gaussian MRF

- Real-valued attributes $\mathbf{a}_u = [\mathbf{x}_u; y_u] \in \mathbb{R}^{p+1}$ on each node u .
- $A_i = i$ th attribute over all nodes.
- $\mathbf{N} = \mathbf{I} - \mathbf{D}^{-1/2} \mathbf{W} \mathbf{D}^{-1/2}$ is the normalized Laplacian.
- Gaussian MRF parameters: $\mathbf{H} \succcurlyeq 0 \in \mathbb{R}^{(p+1) \times (p+1)}$, $\mathbf{h} \in \mathbb{R}_+^{p+1}$

STATISTICAL MODEL FOR NODE ATTRIBUTES

The joint distribution of all node attributes is a Gaussian MRF

- Real-valued attributes $\mathbf{a}_u = [\mathbf{x}_u; y_u] \in \mathbb{R}^{p+1}$ on each node u .
- $A_i = i$ th attribute over all nodes.
- $\mathbf{N} = \mathbf{I} - \mathbf{D}^{-1/2} \mathbf{W} \mathbf{D}^{-1/2}$ is the normalized Laplacian.
- Gaussian MRF parameters: $\mathbf{H} \succcurlyeq 0 \in \mathbb{R}^{(p+1) \times (p+1)}$, $\mathbf{h} \in \mathbb{R}_+^{p+1}$
- Gaussian MRF log-potential,

STATISTICAL MODEL FOR NODE ATTRIBUTES

The joint distribution of all node attributes is a Gaussian MRF

- Real-valued attributes $\mathbf{a}_u = [\mathbf{x}_u; y_u] \in \mathbb{R}^{p+1}$ on each node u .
- $\mathbf{A}_i = i$ th attribute over all nodes.
- $\mathbf{N} = \mathbf{I} - \mathbf{D}^{-1/2} \mathbf{W} \mathbf{D}^{-1/2}$ is the normalized Laplacian.
- Gaussian MRF parameters: $\mathbf{H} \succcurlyeq 0 \in \mathbb{R}^{(p+1) \times (p+1)}$, $\mathbf{h} \in \mathbb{R}_+^{p+1}$
- Gaussian MRF log-potential,

$$\varphi(\mathbf{A}|\mathbf{H}, \mathbf{h}) = -\frac{1}{2} \sum_{u=1}^n \mathbf{a}_u^\top \mathbf{H} \mathbf{a}_u + \frac{1}{2} \sum_{i=1}^{p+1} h_i \mathbf{A}_i^\top \mathbf{N} \mathbf{A}_i$$

STATISTICAL MODEL FOR NODE ATTRIBUTES

The joint distribution of all node attributes is a Gaussian MRF

- Real-valued attributes $\mathbf{a}_u = [\mathbf{x}_u; y_u] \in \mathbb{R}^{p+1}$ on each node u .
- $\mathbf{A}_i = i$ th attribute over all nodes.
- $\mathbf{N} = \mathbf{I} - \mathbf{D}^{-1/2} \mathbf{W} \mathbf{D}^{-1/2}$ is the normalized Laplacian.
- Gaussian MRF parameters: $\mathbf{H} \succcurlyeq 0 \in \mathbb{R}^{(p+1) \times (p+1)}$, $\mathbf{h} \in \mathbb{R}_+^{p+1}$
- Gaussian MRF log-potential,

$$\varphi(\mathbf{A}|\mathbf{H}, \mathbf{h}) = \underbrace{\frac{1}{2} \sum_{u=1}^n \mathbf{a}_u^\top \mathbf{H} \mathbf{a}_u}_{\text{correlated attributes on each node } u} + \frac{1}{2} \sum_{i=1}^{p+1} h_i \mathbf{A}_i^\top \mathbf{N} \mathbf{A}_i$$

STATISTICAL MODEL FOR NODE ATTRIBUTES

The joint distribution of all node attributes is a Gaussian MRF

- Real-valued attributes $\mathbf{a}_u = [\mathbf{x}_u; y_u] \in \mathbb{R}^{p+1}$ on each node u .
- $\mathbf{A}_i = i$ th attribute over all nodes.
- $\mathbf{N} = \mathbf{I} - \mathbf{D}^{-1/2} \mathbf{W} \mathbf{D}^{-1/2}$ is the normalized Laplacian.
- Gaussian MRF parameters: $\mathbf{H} \succcurlyeq 0 \in \mathbb{R}^{(p+1) \times (p+1)}$, $\mathbf{h} \in \mathbb{R}_+^{p+1}$
- Gaussian MRF log-potential, with $\mathbf{A}_i^\top \mathbf{N} \mathbf{A}_i = \sum_{(u,v) \in E} (A_{ui}/\sqrt{d_u} - A_{vi}/\sqrt{d_v})^2$

$$\varphi(\mathbf{A}|\mathbf{H}, \mathbf{h}) = \underbrace{\frac{1}{2} \sum_{u=1}^n \mathbf{a}_u^\top \mathbf{H} \mathbf{a}_u}_{\text{correlated attributes on each node } u} + \frac{1}{2} \sum_{i=1}^{p+1} h_i \mathbf{A}_i^\top \mathbf{N} \mathbf{A}_i$$

STATISTICAL MODEL FOR NODE ATTRIBUTES

The joint distribution of all node attributes is a Gaussian MRF

- Real-valued attributes $\mathbf{a}_u = [\mathbf{x}_u; y_u] \in \mathbb{R}^{p+1}$ on each node u .
- $\mathbf{A}_i = i$ th attribute over all nodes.
- $\mathbf{N} = \mathbf{I} - \mathbf{D}^{-1/2} \mathbf{W} \mathbf{D}^{-1/2}$ is the normalized Laplacian.
- Gaussian MRF parameters: $\mathbf{H} \succcurlyeq 0 \in \mathbb{R}^{(p+1) \times (p+1)}$, $\mathbf{h} \in \mathbb{R}_+^{p+1}$
- Gaussian MRF log-potential, with $\mathbf{A}_i^\top \mathbf{N} \mathbf{A}_i = \sum_{(u,v) \in E} (A_{ui}/\sqrt{d_u} - A_{vi}/\sqrt{d_v})^2$

$$\varphi(\mathbf{A}|\mathbf{H}, \mathbf{h}) = \underbrace{\frac{1}{2} \sum_{u=1}^n \mathbf{a}_u^\top \mathbf{H} \mathbf{a}_u}_{\text{correlated attributes on each node } u} + \underbrace{\frac{1}{2} \sum_{i=1}^{p+1} h_i \mathbf{A}_i^\top \mathbf{N} \mathbf{A}_i}_{\text{smoothness for each attribute } i}$$

STATISTICAL MODEL FOR NODE ATTRIBUTES

The joint distribution of all node attributes is a Gaussian MRF

- Real-valued attributes $\mathbf{a}_u = [\mathbf{x}_u; y_u] \in \mathbb{R}^{p+1}$ on each node u .
- $\mathbf{A}_i = i$ th attribute over all nodes.
- $\mathbf{N} = \mathbf{I} - \mathbf{D}^{-1/2} \mathbf{W} \mathbf{D}^{-1/2}$ is the normalized Laplacian.
- Gaussian MRF parameters: $\mathbf{H} \succcurlyeq 0 \in \mathbb{R}^{(p+1) \times (p+1)}$, $\mathbf{h} \in \mathbb{R}_+^{p+1}$
- Gaussian MRF log-potential, with $\mathbf{A}_i^\top \mathbf{N} \mathbf{A}_i = \sum_{(u,v) \in E} (A_{ui}/\sqrt{d_u} - A_{vi}/\sqrt{d_v})^2$

$$\varphi(\mathbf{A}|\mathbf{H}, \mathbf{h}) = \underbrace{\frac{1}{2} \sum_{u=1}^n \mathbf{a}_u^\top \mathbf{H} \mathbf{a}_u}_{\text{correlated attributes on each node } u} + \underbrace{\frac{1}{2} \sum_{i=1}^{p+1} h_i \mathbf{A}_i^\top \mathbf{N} \mathbf{A}_i}_{\text{smoothness for each attribute } i}$$

- Probability density function: $\rho(\mathbf{A}|\mathbf{H}, \mathbf{h}) = \frac{e^{-\varphi(\mathbf{A}|\mathbf{H}, \mathbf{h})}}{\int d\mathbf{A}' e^{-\varphi(\mathbf{A}'|\mathbf{H}, \mathbf{h})}}$

STATISTICAL MODEL FOR NODE ATTRIBUTES

The joint distribution of all node attributes is a Gaussian MRF

- Real-valued attributes $\mathbf{a}_u = [\mathbf{x}_u; y_u] \in \mathbb{R}^{p+1}$ on each node u .
- $\mathbf{A}_i = i$ th attribute over all nodes.
- $\mathbf{N} = \mathbf{I} - \mathbf{D}^{-1/2} \mathbf{W} \mathbf{D}^{-1/2}$ is the normalized Laplacian.
- Gaussian MRF parameters: $\mathbf{H} \succcurlyeq 0 \in \mathbb{R}^{(p+1) \times (p+1)}$, $\mathbf{h} \in \mathbb{R}_+^{p+1}$
- Gaussian MRF log-potential, with $\mathbf{A}_i^\top \mathbf{N} \mathbf{A}_i = \sum_{(u,v) \in E} (A_{ui}/\sqrt{d_u} - A_{vi}/\sqrt{d_v})^2$

$$\varphi(\mathbf{A}|\mathbf{H}, \mathbf{h}) = \underbrace{\frac{1}{2} \sum_{u=1}^n \mathbf{a}_u^\top \mathbf{H} \mathbf{a}_u}_{\text{correlated attributes on each node } u} + \underbrace{\frac{1}{2} \sum_{i=1}^{p+1} h_i \mathbf{A}_i^\top \mathbf{N} \mathbf{A}_i}_{\text{smoothness for each attribute } i}$$

- Probability density function: $\rho(\mathbf{A}|\mathbf{H}, \mathbf{h}) = \frac{e^{-\varphi(\mathbf{A}|\mathbf{H}, \mathbf{h})}}{\int d\mathbf{A}' e^{-\varphi(\mathbf{A}'|\mathbf{H}, \mathbf{h})}}$
- Just a multivariate Gaussian: $\text{vec}(\mathbf{A}) \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Gamma}^{-1})$, $\boldsymbol{\Gamma} = \mathbf{H} \otimes \mathbf{I}_n + \text{diag}(\mathbf{h}) \otimes \mathbf{N}$

CASE 1. LINEAR REGRESSION IF NO EDGE EXISTS

This is a well-known result for standard linear models

- Gaussian MRF log-potential simplifies as:

$$\varphi(\mathbf{A}|\mathbf{H}, \mathbf{h}) = \frac{1}{2} \sum_{u=1}^n \mathbf{a}_u^\top \mathbf{H} \mathbf{a}_u + \frac{1}{2} \sum_{i=1}^{p+1} h_i \mathbf{A}_i^\top \mathbf{N} \mathbf{A}_i$$

CASE 1. LINEAR REGRESSION IF NO EDGE EXISTS

This is a well-known result for standard linear models

- Gaussian MRF log-potential simplifies as:

$$\varphi(\mathbf{A}|\mathbf{H}, \mathbf{h}) = \frac{1}{2} \sum_{u=1}^n \mathbf{a}_u^\top \mathbf{H} \mathbf{a}_u + \frac{1}{2} \sum_{i=1}^{p+1} h_i \mathbf{A}_i^\top \mathbf{N} \mathbf{A}_i$$

CASE 1. LINEAR REGRESSION IF NO EDGE EXISTS

This is a well-known result for standard linear models

- Gaussian MRF log-potential simplifies as:

$$\varphi(\mathbf{A}|\mathbf{H}, \mathbf{h}) = \frac{1}{2} \sum_{u=1}^n \mathbf{a}_u^\top \mathbf{H} \mathbf{a}_u + \frac{1}{2} \sum_{i=1}^{p+1} h_i \mathbf{A}_i^\top \mathbf{N} \mathbf{A}_i$$

- Probability density function can be factorized as:

$$\rho(\mathbf{A}|\mathbf{H}, \mathbf{h}) = \prod_{u=1}^n \frac{e^{-\frac{1}{2} \mathbf{a}_u^\top \mathbf{H} \mathbf{a}_u}}{\int d\mathbf{a}'_u e^{-\frac{1}{2} \mathbf{a}'_u^\top \mathbf{H} \mathbf{a}'_u}} = \prod_{u=1}^n \sqrt{\frac{\det(\mathbf{H})}{(2\pi)^{q+1}}} \cdot e^{-\frac{1}{2} \mathbf{a}_u^\top \mathbf{H} \mathbf{a}_u}$$

CASE 1. LINEAR REGRESSION IF NO EDGE EXISTS

This is a well-known result for standard linear models

- Gaussian MRF log-potential simplifies as:

$$\varphi(\mathbf{A}|\mathbf{H}, \mathbf{h}) = \frac{1}{2} \sum_{u=1}^n \mathbf{a}_u^\top \mathbf{H} \mathbf{a}_u + \frac{1}{2} \sum_{i=1}^{p+1} h_i \mathbf{A}_i^\top \mathbf{N} \mathbf{A}_i$$

- Probability density function can be factorized as:

$$\rho(\mathbf{A}|\mathbf{H}, \mathbf{h}) = \prod_{u=1}^n \frac{e^{-\frac{1}{2} \mathbf{a}_u^\top \mathbf{H} \mathbf{a}_u}}{\int d\mathbf{a}'_u e^{-\frac{1}{2} \mathbf{a}'_u^\top \mathbf{H} \mathbf{a}'_u}} = \prod_{u=1}^n \sqrt{\frac{\det(\mathbf{H})}{(2\pi)^{q+1}}} \cdot e^{-\frac{1}{2} \mathbf{a}_u^\top \mathbf{H} \mathbf{a}_u}$$

- Predict \mathbf{y}_U with its expectation conditioned on the **features**:

$$E[y_u | \mathbf{X} = \mathbf{X}] = E[y_u | \mathbf{x}_u = \mathbf{x}_u] = \mathbf{x}_u^\top (-\mathbf{H}_{1:p,p+1}/H_{p+1,p+1}) = \mathbf{x}_u^\top \beta$$

CASE 1. LINEAR REGRESSION IF NO EDGE EXISTS

This is a well-known result for standard linear models

- Gaussian MRF log-potential simplifies as:

$$\varphi(\mathbf{A}|\mathbf{H}, \mathbf{h}) = \frac{1}{2} \sum_{u=1}^n \mathbf{a}_u^\top \mathbf{H} \mathbf{a}_u + \frac{1}{2} \sum_{i=1}^{p+1} h_i \mathbf{A}_i^\top \mathbf{N} \mathbf{A}_i$$

- Probability density function can be factorized as:

$$\rho(\mathbf{A}|\mathbf{H}, \mathbf{h}) = \prod_{u=1}^n \frac{e^{-\frac{1}{2} \mathbf{a}_u^\top \mathbf{H} \mathbf{a}_u}}{\int d\mathbf{a}'_u e^{-\frac{1}{2} \mathbf{a}'_u^\top \mathbf{H} \mathbf{a}'_u}} = \prod_{u=1}^n \sqrt{\frac{\det(\mathbf{H})}{(2\pi)^{q+1}}} \cdot e^{-\frac{1}{2} \mathbf{a}_u^\top \mathbf{H} \mathbf{a}_u}$$

- Predict \mathbf{y}_U with its expectation conditioned on the **features**:

$$E[y_u | \mathbf{X} = \mathbf{X}] = E[y_u | \mathbf{x}_u = \mathbf{x}_u] = \mathbf{x}_u^\top (-\mathbf{H}_{1:p,p+1}/H_{p+1,p+1}) = \mathbf{x}_u^\top \beta$$

β is learned directly with OLS in a **discriminative** manner on the training nodes

CASE 2. LABEL PROPAGATION IF NO FEATURE

- Model parameters \mathbf{H}, \mathbf{h} reduce to positive scalars:

$$\text{vec}(\mathbf{A}) \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Gamma}^{-1}), \quad \boldsymbol{\Gamma} = \mathbf{H} \otimes \mathbf{I}_n + \text{diag}(\mathbf{h}) \otimes \mathbf{N}$$

In practice, ω or α can be tuned with cross-validation on the training set.

CASE 2. LABEL PROPAGATION IF NO FEATURE

- Model parameters H, h reduce to positive scalars:

$$\text{vec}(\mathbf{A}) \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Gamma}^{-1}), \quad \boldsymbol{\Gamma} = \cancel{H} \otimes \mathbf{I}_n + \text{diag}(\cancel{h}) \otimes \mathbf{N}$$

In practice, ω or α can be tuned with cross-validation on the training set.

CASE 2. LABEL PROPAGATION IF NO FEATURE

- Model parameters \mathbf{H}, \mathbf{h} reduce to positive scalars:

$$\mathbf{y} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Gamma}^{-1}), \quad \boldsymbol{\Gamma} = \textcolor{red}{H}\mathbf{I}_n + \textcolor{blue}{h}\mathbf{N}$$

In practice, ω or α can be tuned with cross-validation on the training set.

CASE 2. LABEL PROPAGATION IF NO FEATURE

- Model parameters H, h reduce to positive scalars:

$$\mathbf{y} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Gamma}^{-1}), \quad \boldsymbol{\Gamma} = \textcolor{red}{H}\mathbf{I}_n + \textcolor{blue}{h}\mathbf{N}$$

H controls noise: if no edge exists, H^{-1} is variance on vertex;

In practice, ω or α can be tuned with cross-validation on the training set.

CASE 2. LABEL PROPAGATION IF NO FEATURE

- Model parameters H, h reduce to positive scalars:

$$\mathbf{y} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Gamma}^{-1}), \quad \boldsymbol{\Gamma} = \textcolor{red}{H}\mathbf{I}_n + \textcolor{blue}{h}\mathbf{N}$$

H controls noise: if no edge exists, H^{-1} is variance on vertex; h controls smoothness

In practice, ω or α can be tuned with cross-validation on the training set.

CASE 2. LABEL PROPAGATION IF NO FEATURE

- Model parameters H, h reduce to positive scalars:

$$\mathbf{y} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Gamma}^{-1}), \quad \boldsymbol{\Gamma} = \textcolor{red}{H}\mathbf{I}_n + \textcolor{blue}{h}\mathbf{N}$$

H controls noise: if no edge exists, H^{-1} is variance on vertex; h controls smoothness

- Predict \mathbf{y}_U with its expectation conditioned on the **training labels**:

In practice, ω or α can be tuned with cross-validation on the training set.

CASE 2. LABEL PROPAGATION IF NO FEATURE

- Model parameters \mathbf{H}, \mathbf{h} reduce to positive scalars:

$$\mathbf{y} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Gamma}^{-1}), \quad \boldsymbol{\Gamma} = \textcolor{red}{H}\mathbf{I}_n + \textcolor{blue}{h}\mathbf{N}$$

H controls noise: if no edge exists, H^{-1} is variance on vertex; h controls smoothness

- Predict \mathbf{y}_U with its expectation conditioned on the **training labels**:

$$\begin{aligned} E[\mathbf{y}_U | \mathbf{y}_L = \mathbf{y}_L] &= -\boldsymbol{\Gamma}_{UU}^{-1} \boldsymbol{\Gamma}_{UL} \mathbf{y}_L = -(H\mathbf{I}_n + h\mathbf{N})_{UU}^{-1} (H\mathbf{I}_n + h\mathbf{N})_{UL} \mathbf{y}_L \\ &= -(\mathbf{I}_n + \omega\mathbf{N})_{UU}^{-1} (\mathbf{I}_n + \omega\mathbf{N})_{UL} \mathbf{y}_L \end{aligned}$$

$$\omega = \textcolor{blue}{h}/\textcolor{red}{H};$$

In practice, ω or α can be tuned with cross-validation on the training set.

CASE 2. LABEL PROPAGATION IF NO FEATURE

- Model parameters \mathbf{H}, \mathbf{h} reduce to positive scalars:

$$\mathbf{y} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Gamma}^{-1}), \quad \boldsymbol{\Gamma} = \textcolor{red}{H}\mathbf{I}_n + \textcolor{blue}{h}\mathbf{N}$$

H controls noise: if no edge exists, H^{-1} is variance on vertex; h controls smoothness

- Predict \mathbf{y}_U with its expectation conditioned on the **training labels**:

$$\begin{aligned} E[\mathbf{y}_U | \mathbf{y}_L = \mathbf{y}_L] &= -\boldsymbol{\Gamma}_{UU}^{-1} \boldsymbol{\Gamma}_{UL} \mathbf{y}_L = -(H\mathbf{I}_n + h\mathbf{N})_{UU}^{-1} (H\mathbf{I}_n + h\mathbf{N})_{UL} \mathbf{y}_L \\ &= -(\mathbf{I}_n + \omega\mathbf{N})_{UU}^{-1} (\mathbf{I}_n + \omega\mathbf{N})_{UL} \mathbf{y}_L \end{aligned}$$

$\omega = h/H$; The conditional expectation is the solution to a constrained LP:

In practice, ω or α can be tuned with cross-validation on the training set.

CASE 2. LABEL PROPAGATION IF NO FEATURE

- Model parameters \mathbf{H}, \mathbf{h} reduce to positive scalars:

$$\mathbf{y} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Gamma}^{-1}), \quad \boldsymbol{\Gamma} = \textcolor{red}{H}\mathbf{I}_n + \textcolor{blue}{h}\mathbf{N}$$

H controls noise: if no edge exists, H^{-1} is variance on vertex; h controls smoothness

- Predict \mathbf{y}_U with its expectation conditioned on the **training labels**:

$$\begin{aligned} E[\mathbf{y}_U | \mathbf{y}_L = \mathbf{y}_L] &= -\boldsymbol{\Gamma}_{UU}^{-1} \boldsymbol{\Gamma}_{UL} \mathbf{y}_L = -(H\mathbf{I}_n + h\mathbf{N})_{UU}^{-1} (H\mathbf{I}_n + h\mathbf{N})_{UL} \mathbf{y}_L \\ &= -(\mathbf{I}_n + \omega\mathbf{N})_{UU}^{-1} (\mathbf{I}_n + \omega\mathbf{N})_{UL} \mathbf{y}_L \end{aligned}$$

$\omega = h/H$; The conditional expectation is the solution to a constrained LP:

$$\mathbf{y}_U^{(k)} = [\alpha \cdot \mathbf{S} \mathbf{y}^{(k-1)} + (1 - \alpha) \cdot \hat{\mathbf{y}}]_U ; \quad \mathbf{y}_L^{(k)} = \mathbf{y}_L$$

In practice, ω or α can be tuned with cross-validation on the training set.

CASE 2. LABEL PROPAGATION IF NO FEATURE

- Model parameters \mathbf{H}, \mathbf{h} reduce to positive scalars:

$$\mathbf{y} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Gamma}^{-1}), \quad \boldsymbol{\Gamma} = \textcolor{red}{H}\mathbf{I}_n + \textcolor{blue}{h}\mathbf{N}$$

H controls noise: if no edge exists, H^{-1} is variance on vertex; h controls smoothness

- Predict \mathbf{y}_U with its expectation conditioned on the **training labels**:

$$\begin{aligned} E[\mathbf{y}_U | \mathbf{y}_L = \mathbf{y}_L] &= -\boldsymbol{\Gamma}_{UU}^{-1} \boldsymbol{\Gamma}_{UL} \mathbf{y}_L = -(H\mathbf{I}_n + h\mathbf{N})_{UU}^{-1} (H\mathbf{I}_n + h\mathbf{N})_{UL} \mathbf{y}_L \\ &= -(\mathbf{I}_n + \omega\mathbf{N})_{UU}^{-1} (\mathbf{I}_n + \omega\mathbf{N})_{UL} \mathbf{y}_L \end{aligned}$$

$\omega = h/H$; The conditional expectation is the solution to a constrained LP:

$$\mathbf{y}_U^{(k)} = [\alpha \cdot \mathbf{S} \mathbf{y}^{(k-1)} + (1-\alpha) \cdot \hat{\mathbf{y}}]_U; \quad \mathbf{y}_L^{(k)} = \mathbf{y}_L$$

$\alpha = \omega/(1+\omega) \in (0, 1)$ controls smoothing level

In practice, ω or α can be tuned with cross-validation on the training set.

CASE 2. LABEL PROPAGATION IF NO FEATURE

- Model parameters \mathbf{H}, \mathbf{h} reduce to positive scalars:

$$\mathbf{y} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Gamma}^{-1}), \quad \boldsymbol{\Gamma} = \textcolor{red}{H}\mathbf{I}_n + \textcolor{blue}{h}\mathbf{N}$$

H controls noise: if no edge exists, H^{-1} is variance on vertex; h controls smoothness

- Predict \mathbf{y}_U with its expectation conditioned on the **training labels**:

$$\begin{aligned} E[\mathbf{y}_U | \mathbf{y}_L = \mathbf{y}_L] &= -\boldsymbol{\Gamma}_{UU}^{-1} \boldsymbol{\Gamma}_{UL} \mathbf{y}_L = -(H\mathbf{I}_n + h\mathbf{N})_{UU}^{-1} (H\mathbf{I}_n + h\mathbf{N})_{UL} \mathbf{y}_L \\ &= -(\mathbf{I}_n + \omega\mathbf{N})_{UU}^{-1} (\mathbf{I}_n + \omega\mathbf{N})_{UL} \mathbf{y}_L \end{aligned}$$

$\omega = h/H$; The conditional expectation is the solution to a constrained LP:

$$\mathbf{y}_U^{(k)} = [\alpha \cdot \mathbf{S} \mathbf{y}^{(k-1)} + (1-\alpha) \cdot \hat{\mathbf{y}}]_U; \quad \mathbf{y}_L^{(k)} = \mathbf{y}_L$$

$\alpha = \omega/(1+\omega) \in (0, 1)$ controls smoothing level

- high smoothness $\rightarrow h \uparrow \rightarrow \omega \uparrow \rightarrow \alpha \uparrow$;

In practice, ω or α can be tuned with cross-validation on the training set.

CASE 2. LABEL PROPAGATION IF NO FEATURE

- Model parameters \mathbf{H}, \mathbf{h} reduce to positive scalars:

$$\mathbf{y} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Gamma}^{-1}), \quad \boldsymbol{\Gamma} = \textcolor{red}{H}\mathbf{I}_n + \textcolor{blue}{h}\mathbf{N}$$

H controls noise: if no edge exists, H^{-1} is variance on vertex; h controls smoothness

- Predict \mathbf{y}_U with its expectation conditioned on the **training labels**:

$$\begin{aligned} E[\mathbf{y}_U | \mathbf{y}_L = \mathbf{y}_L] &= -\boldsymbol{\Gamma}_{UU}^{-1} \boldsymbol{\Gamma}_{UL} \mathbf{y}_L = -(H\mathbf{I}_n + h\mathbf{N})_{UU}^{-1} (H\mathbf{I}_n + h\mathbf{N})_{UL} \mathbf{y}_L \\ &= -(\mathbf{I}_n + \omega\mathbf{N})_{UU}^{-1} (\mathbf{I}_n + \omega\mathbf{N})_{UL} \mathbf{y}_L \end{aligned}$$

$\omega = h/H$; The conditional expectation is the solution to a constrained LP:

$$\mathbf{y}_U^{(k)} = [\alpha \cdot \mathbf{S} \mathbf{y}^{(k-1)} + (1-\alpha) \cdot \hat{\mathbf{y}}]_U; \quad \mathbf{y}_L^{(k)} = \mathbf{y}_L$$

$\alpha = \omega/(1+\omega) \in (0, 1)$ controls smoothing level

- high smoothness $\rightarrow h \uparrow \rightarrow \omega \uparrow \rightarrow \alpha \uparrow$; high noise $\rightarrow H \downarrow \rightarrow \omega \uparrow \rightarrow \alpha \uparrow$

In practice, ω or α can be tuned with cross-validation on the training set.

CASE 3. LINEAR GRAPH CONVOLUTION

Computes expectation of y in the full model conditioned only on features

- The joint distribution of all attributes is given by:

$$\text{vec}(\mathbf{A}) \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Gamma}^{-1}), \quad \boldsymbol{\Gamma} = \mathbf{H} \otimes \mathbf{I}_n + \text{diag}(\mathbf{h}) \otimes \mathbf{N}$$

CASE 3. LINEAR GRAPH CONVOLUTION

Computes expectation of y in the full model conditioned only on features

- The joint distribution of all attributes is given by:

$$\text{vec}(\mathbf{A}) \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Gamma}^{-1}), \quad \boldsymbol{\Gamma} = \mathbf{H} \otimes \mathbf{I}_n + \text{diag}(\mathbf{h}) \otimes \mathbf{N}$$

- Predict y with its expectation conditioned on the **features**:

CASE 3. LINEAR GRAPH CONVOLUTION

Computes expectation of \mathbf{y} in the full model conditioned only on features

- The joint distribution of all attributes is given by:

$$\text{vec}(\mathbf{A}) \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Gamma}^{-1}), \quad \boldsymbol{\Gamma} = \mathbf{H} \otimes \mathbf{I}_n + \text{diag}(\mathbf{h}) \otimes \mathbf{N}$$

- Predict \mathbf{y} with its expectation conditioned on the **features**:

$$\bar{\mathbf{y}} = E[\mathbf{y} | \mathbf{X} = \mathbf{X}] = (H_{p+1,p+1} \mathbf{I}_n + h_{p+1} \mathbf{N})^{-1} \left(-\mathbf{H}_{1:p,p+1}^\top \otimes \mathbf{I}_n \right) \text{vec}(\mathbf{X})$$

CASE 3. LINEAR GRAPH CONVOLUTION

Computes expectation of \mathbf{y} in the full model conditioned only on features

- The joint distribution of all attributes is given by:

$$\text{vec}(\mathbf{A}) \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Gamma}^{-1}), \quad \boldsymbol{\Gamma} = \mathbf{H} \otimes \mathbf{I}_n + \text{diag}(\mathbf{h}) \otimes \mathbf{N}$$

- Predict \mathbf{y} with its expectation conditioned on the **features**:

$$\begin{aligned}\bar{\mathbf{y}} &= E[\mathbf{y} | \mathbf{X} = \mathbf{X}] = (H_{p+1,p+1} \mathbf{I}_n + h_{p+1} \mathbf{N})^{-1} \left(-\mathbf{H}_{1:p,p+1}^\top \otimes \mathbf{I}_n \right) \text{vec}(\mathbf{X}) \\ &= (\mathbf{I}_n + \omega \mathbf{N})^{-1} \mathbf{X} \beta\end{aligned}$$

CASE 3. LINEAR GRAPH CONVOLUTION

Computes expectation of \mathbf{y} in the full model conditioned only on features

- The joint distribution of all attributes is given by:

$$\text{vec}(\mathbf{A}) \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Gamma}^{-1}), \quad \boldsymbol{\Gamma} = \mathbf{H} \otimes \mathbf{I}_n + \text{diag}(\mathbf{h}) \otimes \mathbf{N}$$

- Predict \mathbf{y} with its expectation conditioned on the **features**:

$$\begin{aligned}\bar{\mathbf{y}} &= E[\mathbf{y} | \mathbf{X} = \mathbf{X}] = (\mathbf{H}_{p+1,p+1} \mathbf{I}_n + \mathbf{h}_{p+1} \mathbf{N})^{-1} \left(-\mathbf{H}_{1:p,p+1}^\top \otimes \mathbf{I}_n \right) \text{vec}(\mathbf{X}) \\ &= \underbrace{(\mathbf{I}_n + \omega \mathbf{N})^{-1} \mathbf{X} \beta}_{\text{LP on features!}}\end{aligned}$$

$(\mathbf{I}_n + \omega \mathbf{N})^{-1} \mathbf{X}$ is the solution to label propagation for the features:

CASE 3. LINEAR GRAPH CONVOLUTION

Computes expectation of \mathbf{y} in the full model conditioned only on features

- The joint distribution of all attributes is given by:

$$\text{vec}(\mathbf{A}) \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Gamma}^{-1}), \quad \boldsymbol{\Gamma} = \mathbf{H} \otimes \mathbf{I}_n + \text{diag}(\mathbf{h}) \otimes \mathbf{N}$$

- Predict \mathbf{y} with its expectation conditioned on the **features**:

$$\begin{aligned}\bar{\mathbf{y}} &= E[\mathbf{y} | \mathbf{X} = \mathbf{X}] = (\mathbf{H}_{p+1,p+1} \mathbf{I}_n + \mathbf{h}_{p+1} \mathbf{N})^{-1} \left(-\mathbf{H}_{1:p,p+1}^\top \otimes \mathbf{I}_n \right) \text{vec}(\mathbf{X}) \\ &= \underbrace{(\mathbf{I}_n + \omega \mathbf{N})^{-1} \mathbf{X} \beta}_{\text{LP on features!}}\end{aligned}$$

$(\mathbf{I}_n + \omega \mathbf{N})^{-1} \mathbf{X}$ is the solution to label propagation for the features:

$$\mathbf{X}^{(k)} = \alpha \cdot \mathbf{S} \mathbf{X}^{(k-1)} + (1 - \alpha) \cdot \mathbf{X}$$

CASE 3. LINEAR GRAPH CONVOLUTION

Computes expectation of \mathbf{y} in the full model conditioned only on features

- The joint distribution of all attributes is given by:

$$\text{vec}(\mathbf{A}) \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Gamma}^{-1}), \quad \boldsymbol{\Gamma} = \mathbf{H} \otimes \mathbf{I}_n + \text{diag}(\mathbf{h}) \otimes \mathbf{N}$$

- Predict \mathbf{y} with its expectation conditioned on the **features**:

$$\begin{aligned}\bar{\mathbf{y}} &= E[\mathbf{y} | \mathbf{X} = \mathbf{X}] = (H_{p+1,p+1} \mathbf{I}_n + h_{p+1} \mathbf{N})^{-1} \left(-\mathbf{H}_{1:p,p+1}^\top \otimes \mathbf{I}_n \right) \text{vec}(\mathbf{X}) \\ &= \underbrace{(\mathbf{I}_n + \omega \mathbf{N})^{-1} \mathbf{X} \beta}_{\text{LP on features!}}\end{aligned}$$

$(\mathbf{I}_n + \omega \mathbf{N})^{-1} \mathbf{X}$ is the solution to label propagation for the features:

$$\mathbf{X}^{(k)} = \alpha \cdot \mathbf{S} \mathbf{X}^{(k-1)} + (1 - \alpha) \cdot \mathbf{X}$$

- Linear graph convolution (LGC)

CASE 3. LINEAR GRAPH CONVOLUTION

Computes expectation of \mathbf{y} in the full model conditioned only on features

- The joint distribution of all attributes is given by:

$$\text{vec}(\mathbf{A}) \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Gamma}^{-1}), \quad \boldsymbol{\Gamma} = \mathbf{H} \otimes \mathbf{I}_n + \text{diag}(\mathbf{h}) \otimes \mathbf{N}$$

- Predict \mathbf{y} with its expectation conditioned on the **features**:

$$\begin{aligned}\bar{\mathbf{y}} &= E[\mathbf{y} | \mathbf{X} = \mathbf{X}] = (\mathbf{H}_{p+1,p+1} \mathbf{I}_n + \mathbf{h}_{p+1} \mathbf{N})^{-1} \left(-\mathbf{H}_{1:p,p+1}^\top \otimes \mathbf{I}_n \right) \text{vec}(\mathbf{X}) \\ &= \underbrace{(\mathbf{I}_n + \omega \mathbf{N})^{-1} \mathbf{X} \beta}_{\text{LP on features!}}\end{aligned}$$

$(\mathbf{I}_n + \omega \mathbf{N})^{-1} \mathbf{X}$ is the solution to label propagation for the features:

$$\mathbf{X}^{(k)} = \alpha \cdot \mathbf{S} \mathbf{X}^{(k-1)} + (1 - \alpha) \cdot \mathbf{X}$$

- Linear graph convolution (LGC)

- use label propagation to smooth/preprocess features

CASE 3. LINEAR GRAPH CONVOLUTION

Computes expectation of \mathbf{y} in the full model conditioned only on features

- The joint distribution of all attributes is given by:

$$\text{vec}(\mathbf{A}) \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Gamma}^{-1}), \quad \boldsymbol{\Gamma} = \mathbf{H} \otimes \mathbf{I}_n + \text{diag}(\mathbf{h}) \otimes \mathbf{N}$$

- Predict \mathbf{y} with its expectation conditioned on the **features**:

$$\begin{aligned}\bar{\mathbf{y}} &= E[\mathbf{y} | \mathbf{X} = \mathbf{X}] = (\mathbf{H}_{p+1,p+1} \mathbf{I}_n + \mathbf{h}_{p+1} \mathbf{N})^{-1} \left(-\mathbf{H}_{1:p,p+1}^\top \otimes \mathbf{I}_n \right) \text{vec}(\mathbf{X}) \\ &= \underbrace{(\mathbf{I}_n + \omega \mathbf{N})^{-1} \mathbf{X} \beta}_{\text{LP on features!}}\end{aligned}$$

$(\mathbf{I}_n + \omega \mathbf{N})^{-1} \mathbf{X}$ is the solution to label propagation for the features:

$$\mathbf{X}^{(k)} = \alpha \cdot \mathbf{S} \mathbf{X}^{(k-1)} + (1 - \alpha) \cdot \mathbf{X}$$

- Linear graph convolution (LGC)

- use label propagation to smooth/preprocess features
- just like linear regression, learn β with OLS on the training nodes

CASE 3. LINEAR GRAPH CONVOLUTION

$$\begin{array}{ll} \text{LGC} & (1 - \alpha)(\mathbf{I} + \alpha\mathbf{S} + \alpha^2\mathbf{S}^2 + \dots)\mathbf{X}\beta \quad \mathbf{S} = \mathbf{D}^{-1/2}\mathbf{W}\mathbf{D}^{-1/2} \\ [\text{Jia-Benson } 21] & \end{array}$$

$$\begin{array}{ll} \text{SGC} & \tilde{\mathbf{S}}^K \mathbf{X}\beta \quad \tilde{\mathbf{S}} = (\mathbf{D} + \mathbf{I})^{-1/2}(\mathbf{W} + \mathbf{I})(\mathbf{D} + \mathbf{I})^{-1/2} \\ [\text{Wu+ } 19] & \end{array}$$

$$\begin{array}{ll} \text{GCN} & \sigma(\tilde{\mathbf{S}} \dots \sigma(\tilde{\mathbf{S}} \mathbf{X} \Theta^{(1)}) \dots \Theta^{(K)})\beta \\ [\text{Kipf-Welling } 17] & \end{array}$$

- connection to the graph convolutional network:
 - LGC sums decaying contributions from features on increasingly distant neighbors

Neumann expansion, $(\mathbf{I} + \omega\mathbf{N})^{-1} = \left(1 - \frac{\omega}{1+\omega}\right) \left(\mathbf{I} - \frac{\omega}{1+\omega}\mathbf{S}\right)^{-1} = (1 - \alpha)(\mathbf{I} + \alpha\mathbf{S} + \alpha^2\mathbf{S}^2 + \dots)$

CASE 3. LINEAR GRAPH CONVOLUTION

$$\text{LGC} \quad (1 - \alpha)(\mathbf{I} + \alpha\mathbf{S} + \alpha^2\mathbf{S}^2 + \dots)\mathbf{X}\beta \quad \mathbf{S} = \mathbf{D}^{-1/2}\mathbf{W}\mathbf{D}^{-1/2}$$

[Jia-Benson 21]

$$\text{SGC} \quad \tilde{\mathbf{S}}^K \mathbf{X}\beta \quad \tilde{\mathbf{S}} = (\mathbf{D} + \mathbf{I})^{-1/2}(\mathbf{W} + \mathbf{I})(\mathbf{D} + \mathbf{I})^{-1/2}$$

[Wu+ 19]

$$\text{GCN} \quad \sigma(\tilde{\mathbf{S}} \dots \sigma(\tilde{\mathbf{S}} \mathbf{X} \Theta^{(1)}) \dots \Theta^{(K)})\beta$$

[Kipf-Welling 17]

- connection to the graph convolutional network:
 - LGC sums decaying contributions from features on increasingly distant neighbors
 - SGC uses a different smoothing filter $\tilde{\mathbf{S}}^K$ as oppose to $(\mathbf{I} + \omega\mathbf{N})^{-1}$

$$\text{Neumann expansion, } (\mathbf{I} + \omega\mathbf{N})^{-1} = \left(1 - \frac{\omega}{1+\omega}\right) \left(\mathbf{I} - \frac{\omega}{1+\omega}\mathbf{S}\right)^{-1} = (1 - \alpha)(\mathbf{I} + \alpha\mathbf{S} + \alpha^2\mathbf{S}^2 + \dots)$$

CASE 3. LINEAR GRAPH CONVOLUTION

$$\begin{array}{ll} \text{LGC} & (1 - \alpha)(\mathbf{I} + \alpha\mathbf{S} + \alpha^2\mathbf{S}^2 + \dots)\mathbf{X}\beta \quad \mathbf{S} = \mathbf{D}^{-1/2}\mathbf{W}\mathbf{D}^{-1/2} \\ [\text{Jia-Benson } 21] & \end{array}$$

$$\begin{array}{ll} \text{SGC} & \tilde{\mathbf{S}}^K \mathbf{X}\beta \quad \tilde{\mathbf{S}} = (\mathbf{D} + \mathbf{I})^{-1/2}(\mathbf{W} + \mathbf{I})(\mathbf{D} + \mathbf{I})^{-1/2} \\ [\text{Wu+ } 19] & \end{array}$$

$$\begin{array}{ll} \text{GCN} & \sigma(\tilde{\mathbf{S}} \dots \sigma(\tilde{\mathbf{S}} \mathbf{X} \Theta^{(1)}) \dots \Theta^{(K)})\beta \\ [\text{Kipf-Welling } 17] & \end{array}$$

- connection to the graph convolutional network:
 - LGC sums decaying contributions from features on increasingly distant neighbors
 - SGC uses a different smoothing filter $\tilde{\mathbf{S}}^K$ as oppose to $(\mathbf{I} + \omega\mathbf{N})^{-1}$
 - GCN adds nonlinearities to SGC

$$\text{Neumann expansion, } (\mathbf{I} + \omega\mathbf{N})^{-1} = \left(1 - \frac{\omega}{1+\omega}\right) \left(\mathbf{I} - \frac{\omega}{1+\omega}\mathbf{S}\right)^{-1} = (1 - \alpha)(\mathbf{I} + \alpha\mathbf{S} + \alpha^2\mathbf{S}^2 + \dots)$$

CASE 3. LINEAR GRAPH CONVOLUTION

$$\text{LGC} \quad (1 - \alpha)(\mathbf{I} + \alpha\mathbf{S} + \alpha^2\mathbf{S}^2 + \dots)\mathbf{X}\beta \quad \mathbf{S} = \mathbf{D}^{-1/2}\mathbf{W}\mathbf{D}^{-1/2}$$

[Jia-Benson 21]

$$\text{SGC} \quad \tilde{\mathbf{S}}^K \mathbf{X}\beta \quad \tilde{\mathbf{S}} = (\mathbf{D} + \mathbf{I})^{-1/2}(\mathbf{W} + \mathbf{I})(\mathbf{D} + \mathbf{I})^{-1/2}$$

[Wu+ 19]

$$\text{GCN} \quad \sigma(\tilde{\mathbf{S}} \dots \sigma(\tilde{\mathbf{S}} \mathbf{X} \Theta^{(1)}) \dots \Theta^{(K)})\beta$$

[Kipf-Welling 17]

- connection to the graph convolutional network:
 - LGC sums decaying contributions from features on increasingly distant neighbors
 - SGC uses a different smoothing filter $\tilde{\mathbf{S}}^K$ as oppose to $(\mathbf{I} + \omega\mathbf{N})^{-1}$
 - GCN adds nonlinearities to SGC
- research questions:

$$\text{Neumann expansion, } (\mathbf{I} + \omega\mathbf{N})^{-1} = \left(1 - \frac{\omega}{1+\omega}\right) \left(\mathbf{I} - \frac{\omega}{1+\omega}\mathbf{S}\right)^{-1} = (1 - \alpha)(\mathbf{I} + \alpha\mathbf{S} + \alpha^2\mathbf{S}^2 + \dots)$$

CASE 3. LINEAR GRAPH CONVOLUTION

$$\text{LGC} \quad (1 - \alpha)(\mathbf{I} + \alpha\mathbf{S} + \alpha^2\mathbf{S}^2 + \dots)\mathbf{X}\beta \quad \mathbf{S} = \mathbf{D}^{-1/2}\mathbf{W}\mathbf{D}^{-1/2}$$

[Jia-Benson 21]

$$\text{SGC} \quad \tilde{\mathbf{S}}^K \mathbf{X}\beta \quad \tilde{\mathbf{S}} = (\mathbf{D} + \mathbf{I})^{-1/2}(\mathbf{W} + \mathbf{I})(\mathbf{D} + \mathbf{I})^{-1/2}$$

[Wu+ 19]

$$\text{GCN} \quad \sigma(\tilde{\mathbf{S}} \dots \sigma(\tilde{\mathbf{S}} \mathbf{X} \Theta^{(1)}) \dots \Theta^{(K)})\beta$$

[Kipf-Welling 17]

- connection to the graph convolutional network:
 - LGC sums decaying contributions from features on increasingly distant neighbors
 - SGC uses a different smoothing filter $\tilde{\mathbf{S}}^K$ as oppose to $(\mathbf{I} + \omega\mathbf{N})^{-1}$
 - GCN adds nonlinearities to SGC
- research questions:
 - How does SGC and LGC smoothing filters compare?

Neumann expansion, $(\mathbf{I} + \omega\mathbf{N})^{-1} = \left(1 - \frac{\omega}{1+\omega}\right) \left(\mathbf{I} - \frac{\omega}{1+\omega}\mathbf{S}\right)^{-1} = (1 - \alpha)(\mathbf{I} + \alpha\mathbf{S} + \alpha^2\mathbf{S}^2 + \dots)$

CASE 3. LINEAR GRAPH CONVOLUTION

$$\text{LGC} \quad (1 - \alpha)(\mathbf{I} + \alpha\mathbf{S} + \alpha^2\mathbf{S}^2 + \dots)\mathbf{X}\beta \quad \mathbf{S} = \mathbf{D}^{-1/2}\mathbf{W}\mathbf{D}^{-1/2}$$

[Jia-Benson 21]

$$\text{SGC} \quad \tilde{\mathbf{S}}^K \mathbf{X}\beta \quad \tilde{\mathbf{S}} = (\mathbf{D} + \mathbf{I})^{-1/2}(\mathbf{W} + \mathbf{I})(\mathbf{D} + \mathbf{I})^{-1/2}$$

[Wu+ 19]

$$\text{GCN} \quad \sigma(\tilde{\mathbf{S}} \dots \sigma(\tilde{\mathbf{S}} \mathbf{X} \Theta^{(1)}) \dots \Theta^{(K)})\beta$$

[Kipf-Welling 17]

- connection to the graph convolutional network:
 - LGC sums decaying contributions from features on increasingly distant neighbors
 - SGC uses a different smoothing filter $\tilde{\mathbf{S}}^K$ as oppose to $(\mathbf{I} + \omega\mathbf{N})^{-1}$
 - GCN adds nonlinearities to SGC
- research questions:
 - How does SGC and LGC smoothing filters compare?
 - How does nonlinearities in GCN helps performance?

Neumann expansion, $(\mathbf{I} + \omega\mathbf{N})^{-1} = \left(1 - \frac{\omega}{1+\omega}\right) \left(\mathbf{I} - \frac{\omega}{1+\omega}\mathbf{S}\right)^{-1} = (1 - \alpha)(\mathbf{I} + \alpha\mathbf{S} + \alpha^2\mathbf{S}^2 + \dots)$

CASE 3. LINEAR GRAPH CONVOLUTION

$$\text{LGC} \quad (1 - \alpha)(\mathbf{I} + \alpha\mathbf{S} + \alpha^2\mathbf{S}^2 + \dots)\mathbf{X}\beta \quad \mathbf{S} = \mathbf{D}^{-1/2}\mathbf{W}\mathbf{D}^{-1/2}$$

[Jia-Benson 21]

$$\text{SGC} \quad \tilde{\mathbf{S}}^K \mathbf{X}\beta \quad \tilde{\mathbf{S}} = (\mathbf{D} + \mathbf{I})^{-1/2}(\mathbf{W} + \mathbf{I})(\mathbf{D} + \mathbf{I})^{-1/2}$$

[Wu+ 19]

$$\text{GCN} \quad \sigma(\tilde{\mathbf{S}} \dots \sigma(\tilde{\mathbf{S}} \mathbf{X} \Theta^{(1)}) \dots \Theta^{(K)})\beta$$

[Kipf-Welling 17]

- connection to the graph convolutional network:
 - LGC sums decaying contributions from features on increasingly distant neighbors
 - SGC uses a different smoothing filter $\tilde{\mathbf{S}}^K$ as oppose to $(\mathbf{I} + \omega\mathbf{N})^{-1}$
 - GCN adds nonlinearities to SGC
- research questions:
 - How does SGC and LGC smoothing filters compare?
 - How does nonlinearities in GCN helps performance?
- LGC computes $E[\mathbf{y}_U | \mathbf{X}]$,

$$\text{Neumann expansion, } (\mathbf{I} + \omega\mathbf{N})^{-1} = \left(1 - \frac{\omega}{1+\omega}\right) \left(\mathbf{I} - \frac{\omega}{1+\omega}\mathbf{S}\right)^{-1} = (1 - \alpha)(\mathbf{I} + \alpha\mathbf{S} + \alpha^2\mathbf{S}^2 + \dots)$$

CASE 3. LINEAR GRAPH CONVOLUTION

$$\text{LGC} \quad (1 - \alpha)(\mathbf{I} + \alpha\mathbf{S} + \alpha^2\mathbf{S}^2 + \dots)\mathbf{X}\beta \quad \mathbf{S} = \mathbf{D}^{-1/2}\mathbf{W}\mathbf{D}^{-1/2}$$

[Jia-Benson 21]

$$\text{SGC} \quad \tilde{\mathbf{S}}^K \mathbf{X}\beta \quad \tilde{\mathbf{S}} = (\mathbf{D} + \mathbf{I})^{-1/2}(\mathbf{W} + \mathbf{I})(\mathbf{D} + \mathbf{I})^{-1/2}$$

[Wu+ 19]

$$\text{GCN} \quad \sigma(\tilde{\mathbf{S}} \dots \sigma(\tilde{\mathbf{S}} \mathbf{X} \Theta^{(1)}) \dots \Theta^{(K)})\beta$$

[Kipf-Welling 17]

- connection to the graph convolutional network:
 - LGC sums decaying contributions from features on increasingly distant neighbors
 - SGC uses a different smoothing filter $\tilde{\mathbf{S}}^K$ as oppose to $(\mathbf{I} + \omega\mathbf{N})^{-1}$
 - GCN adds nonlinearities to SGC
- research questions:
 - How does SGC and LGC smoothing filters compare?
 - How does nonlinearities in GCN helps performance?
- LGC computes $E[\mathbf{y}_U | \mathbf{X}]$, **why not conditioning on \mathbf{y}_L as well?**

Neumann expansion, $(\mathbf{I} + \omega\mathbf{N})^{-1} = \left(1 - \frac{\omega}{1+\omega}\right) \left(\mathbf{I} - \frac{\omega}{1+\omega}\mathbf{S}\right)^{-1} = (1 - \alpha)(\mathbf{I} + \alpha\mathbf{S} + \alpha^2\mathbf{S}^2 + \dots)$

CASE 4. RESIDUAL PROPAGATION

- The conditional distribution of all labels conditioned on features:

$$\mathbf{y} | \mathbf{X} = \mathbf{X} \sim \mathcal{N}(\bar{\mathbf{y}}, \boldsymbol{\Gamma}_{PP}^{-1}), \quad \boldsymbol{\Gamma}_{PP} = H_{p+1,p+1} \mathbf{I}_n + h_{p+1} \mathbf{N}$$

with the conditional mean given by: $\hat{\mathbf{y}} = (\mathbf{I}_n + \omega \mathbf{N})^{-1} \mathbf{X} \beta$

CASE 4. RESIDUAL PROPAGATION

- The conditional distribution of all labels conditioned on features:

$$\mathbf{y} | \mathbf{X} = \mathbf{X} \sim \mathcal{N}(\bar{\mathbf{y}}, \boldsymbol{\Gamma}_{PP}^{-1}), \quad \boldsymbol{\Gamma}_{PP} = H_{p+1,p+1} \mathbf{I}_n + h_{p+1} \mathbf{N}$$

with the conditional mean given by: $\hat{\mathbf{y}} = (\mathbf{I}_n + \omega \mathbf{N})^{-1} \mathbf{X} \beta$

- Further conditioning on $\mathbf{y}_L = \mathbf{y}_L$

CASE 4. RESIDUAL PROPAGATION

- The conditional distribution of all labels conditioned on features:

$$\mathbf{y} | \mathbf{X} = \mathbf{X} \sim \mathcal{N}(\bar{\mathbf{y}}, \boldsymbol{\Gamma}_{PP}^{-1}), \quad \boldsymbol{\Gamma}_{PP} = H_{p+1,p+1} \mathbf{I}_n + h_{p+1} \mathbf{N}$$

with the conditional mean given by: $\hat{\mathbf{y}} = (\mathbf{I}_n + \omega \mathbf{N})^{-1} \mathbf{X} \beta$

- Further conditioning on $\mathbf{y}_L = \mathbf{y}_L$

$$E[\mathbf{y}_U | \mathbf{X} = \mathbf{X}, \mathbf{y}_L = \mathbf{y}_L] = \hat{\mathbf{y}}_U + (\mathbf{I} + \omega \mathbf{N})_{UU}^{-1} (\mathbf{I} + \omega \mathbf{N})_{UL} (\hat{\mathbf{y}}_L - \mathbf{y}_L)$$

CASE 4. RESIDUAL PROPAGATION

- The conditional distribution of all labels conditioned on features:

$$\mathbf{y} | \mathbf{X} = \mathbf{X} \sim \mathcal{N}(\bar{\mathbf{y}}, \boldsymbol{\Gamma}_{PP}^{-1}), \quad \boldsymbol{\Gamma}_{PP} = H_{p+1,p+1} \mathbf{I}_n + h_{p+1} \mathbf{N}$$

with the conditional mean given by: $\hat{\mathbf{y}} = (\mathbf{I}_n + \omega \mathbf{N})^{-1} \mathbf{X} \beta$

- Further conditioning on $\mathbf{y}_L = \mathbf{y}_L$

$$E[\mathbf{y}_U | \mathbf{X} = \mathbf{X}, \mathbf{y}_L = \mathbf{y}_L] = \hat{\mathbf{y}}_U + \underbrace{(\mathbf{I} + \omega \mathbf{N})_{UU}^{-1} (\mathbf{I} + \omega \mathbf{N})_{UL} (\hat{\mathbf{y}}_L - \mathbf{y}_L)}_{\text{constrained LP from training residuals}}$$

CASE 4. RESIDUAL PROPAGATION

- The conditional distribution of all labels conditioned on features:

$$\mathbf{y} | \mathbf{X} = \mathbf{X} \sim \mathcal{N}(\bar{\mathbf{y}}, \boldsymbol{\Gamma}_{PP}^{-1}), \quad \boldsymbol{\Gamma}_{PP} = H_{p+1,p+1} \mathbf{I}_n + h_{p+1} \mathbf{N}$$

with the conditional mean given by: $\hat{\mathbf{y}} = (\mathbf{I}_n + \omega \mathbf{N})^{-1} \mathbf{X} \beta$

- Further conditioning on $\mathbf{y}_L = \mathbf{y}_L$

$$E[\mathbf{y}_U | \mathbf{X} = \mathbf{X}, \mathbf{y}_L = \mathbf{y}_L] = \hat{\mathbf{y}}_U + \underbrace{(\mathbf{I} + \omega \mathbf{N})_{UU}^{-1} (\mathbf{I} + \omega \mathbf{N})_{UL} (\hat{\mathbf{y}}_L - \mathbf{y}_L)}_{\text{constrained LP from training residuals}}$$

- Linear graph convolution with residual propagation (LGC/RP)

CASE 4. RESIDUAL PROPAGATION

- The conditional distribution of all labels conditioned on features:

$$\mathbf{y} | \mathbf{X} = \mathbf{X} \sim \mathcal{N}(\bar{\mathbf{y}}, \boldsymbol{\Gamma}_{PP}^{-1}), \quad \boldsymbol{\Gamma}_{PP} = H_{p+1,p+1} \mathbf{I}_n + h_{p+1} \mathbf{N}$$

with the conditional mean given by: $\hat{\mathbf{y}} = (\mathbf{I}_n + \omega \mathbf{N})^{-1} \mathbf{X} \beta$

- Further conditioning on $\mathbf{y}_L = \mathbf{y}_L$

$$E[\mathbf{y}_U | \mathbf{X} = \mathbf{X}, \mathbf{y}_L = \mathbf{y}_L] = \hat{\mathbf{y}}_U + \underbrace{(\mathbf{I} + \omega \mathbf{N})_{UU}^{-1} (\mathbf{I} + \omega \mathbf{N})_{UL} (\hat{\mathbf{y}}_L - \mathbf{y}_L)}_{\text{constrained LP from training residuals}}$$

- Linear graph convolution with residual propagation (LGC/RP)
 - train LGC to predict both training nodes and testing nodes
 - run LP to interpolate from training residuals r_L to testing residuals r_U
 - add interpolated testing residuals r_U to initial predictions $\hat{\mathbf{y}}_U$ as corrections

CASE 4. RESIDUAL PROPAGATION

- The conditional distribution of all labels conditioned on features:

$$\mathbf{y} | \mathbf{X} = \mathbf{X} \sim \mathcal{N}(\bar{\mathbf{y}}, \boldsymbol{\Gamma}_{PP}^{-1}), \quad \boldsymbol{\Gamma}_{PP} = H_{p+1,p+1} \mathbf{I}_n + h_{p+1} \mathbf{N}$$

with the conditional mean given by: $\hat{\mathbf{y}} = (\mathbf{I}_n + \omega \mathbf{N})^{-1} \mathbf{X} \beta$

- Further conditioning on $\mathbf{y}_L = \mathbf{y}_L$

$$E[\mathbf{y}_U | \mathbf{X} = \mathbf{X}, \mathbf{y}_L = \mathbf{y}_L] = \hat{\mathbf{y}}_U + \underbrace{(\mathbf{I} + \omega \mathbf{N})_{UU}^{-1} (\mathbf{I} + \omega \mathbf{N})_{UL} (\hat{\mathbf{y}}_L - \mathbf{y}_L)}_{\text{constrained LP from training residuals}}$$

- Linear graph convolution with residual propagation (LGC/RP)
 - train LGC to predict both training nodes and testing nodes
 - run LP to interpolate from training residuals r_L to testing residuals r_U
 - add interpolated testing residuals r_U to initial predictions $\hat{\mathbf{y}}_U$ as corrections
- Residual propagation post-processing is compatible with SGC, GCN, ...

THE DERIVED ALGORITHMS PERFORM SUPER WELL

Dataset	Outcome	LP	LR	LGC (α)	SGC (K)	GCN (K)	LGC/RP	SGC/RP	GCN/RP
U.S.	income	0.40	0.63	0.66 (0.46)	0.51 (1.0)	0.53 (1.3)	0.69	0.55	0.55
	education	0.31	0.71	0.71 (0.00)	0.43 (1.0)	0.47 (1.0)	0.71	0.46	0.48
	unemployment	0.47	0.34	0.39 (0.59)	0.32 (1.3)	0.45 (2.5)	0.54	0.52	0.53
	election	0.52	0.42	0.49 (0.68)	0.43 (1.1)	0.52 (2.1)	0.64	0.61	0.61
CDC	airT	0.95	0.85	0.86 (0.78)	0.86 (2.6)	0.95 (3.0)	0.96	0.97	0.97
	landT	0.89	0.81	0.81 (0.09)	0.79 (1.0)	0.91 (2.4)	0.90	0.93	0.93
	precipitation	0.89	0.59	0.61 (0.93)	0.61 (2.3)	0.79 (3.0)	0.89	0.90	0.90
	sunlight	0.96	0.75	0.81 (0.97)	0.80 (3.0)	0.90 (3.0)	0.96	0.97	0.97
	pm2.5	0.96	0.21	0.27 (0.99)	0.23 (2.7)	0.78 (3.0)	0.96	0.96	0.97
London	income	0.46	0.85	0.85 (0.00)	0.64 (1.0)	0.63 (1.0)	0.85	0.65	0.64
	education	0.65	0.81	0.83 (0.40)	0.74 (1.6)	0.79 (1.4)	0.86	0.77	0.79
	age	0.65	0.73	0.73 (0.17)	0.66 (1.2)	0.70 (1.7)	0.75	0.72	0.72
	election	0.67	0.73	0.81 (0.74)	0.74 (2.0)	0.76 (2.1)	0.85	0.78	0.78
Twitch	days	0.08	0.58	0.59 (0.67)	0.22 (1.4)	0.26 (1.7)	0.60	0.23	0.26

30% training; hyperparameter α , K tuned with cross-validation on training nodes

THE DERIVED ALGORITHMS PERFORM SUPER WELL

Dataset	Outcome	LP	LR	LGC (α)	SGC (K)	GCN (K)	LGC/RP	SGC/RP	GCN/RP
U.S.	income	0.40	0.63	0.66 (0.46)	0.51 (1.0)	0.53 (1.3)	0.69	0.55	0.55
	education	0.31	0.71	0.71 (0.00)	0.43 (1.0)	0.47 (1.0)	0.71	0.46	0.48
	unemployment	0.47	0.34	0.39 (0.59)	0.32 (1.3)	0.45 (2.5)	0.54	0.52	0.53
	election	0.52	0.42	0.49 (0.68)	0.43 (1.1)	0.52 (2.1)	0.64	0.61	0.61
CDC	airT	0.95	0.85	0.86 (0.78)	0.86 (2.6)	0.95 (3.0)	0.96	0.97	0.97
	landT	0.89	0.81	0.81 (0.09)	0.79 (1.0)	0.91 (2.4)	0.90	0.93	0.93
	precipitation	0.89	0.59	0.61 (0.93)	0.61 (2.3)	0.79 (3.0)	0.89	0.90	0.90
	sunlight	0.96	0.75	0.81 (0.97)	0.80 (3.0)	0.90 (3.0)	0.96	0.97	0.97
	pm2.5	0.96	0.21	0.27 (0.99)	0.23 (2.7)	0.78 (3.0)	0.96	0.96	0.97
London	income	0.46	0.85	0.85 (0.00)	0.64 (1.0)	0.63 (1.0)	0.85	0.65	0.64
	education	0.65	0.81	0.83 (0.40)	0.74 (1.6)	0.79 (1.4)	0.86	0.77	0.79
	age	0.65	0.73	0.73 (0.17)	0.66 (1.2)	0.70 (1.7)	0.75	0.72	0.72
	election	0.67	0.73	0.81 (0.74)	0.74 (2.0)	0.76 (2.1)	0.85	0.78	0.78
Twitch	days	0.08	0.58	0.59 (0.67)	0.22 (1.4)	0.26 (1.7)	0.60	0.23	0.26

- LGC consistently outperforms SGC on all datasets (explain from graph filtering later)

30% training; hyperparameter α , K tuned with cross-validation on training nodes

THE DERIVED ALGORITHMS PERFORM SUPER WELL

Dataset	Outcome	LP	LR	LGC (α)	SGC (K)	GCN (K)	LGC/RP	SGC/RP	GCN/RP
U.S.	income	0.40	0.63	0.66 (0.46)	0.51 (1.0)	0.53 (1.3)	0.69	0.55	0.55
	education	0.31	0.71	0.71 (0.00)	0.43 (1.0)	0.47 (1.0)	0.71	0.46	0.48
	unemployment	0.47	0.34	0.39 (0.59)	0.32 (1.3)	0.45 (2.5)	0.54	0.52	0.53
	election	0.52	0.42	0.49 (0.68)	0.43 (1.1)	0.52 (2.1)	0.64	0.61	0.61
CDC	airT	0.95	0.85	0.86 (0.78)	0.86 (2.6)	0.95 (3.0)	0.96	0.97	0.97
	landT	0.89	0.81	0.81 (0.09)	0.79 (1.0)	0.91 (2.4)	0.90	0.93	0.93
	precipitation	0.89	0.59	0.61 (0.93)	0.61 (2.3)	0.79 (3.0)	0.89	0.90	0.90
	sunlight	0.96	0.75	0.81 (0.97)	0.80 (3.0)	0.90 (3.0)	0.96	0.97	0.97
	pm2.5	0.96	0.21	0.27 (0.99)	0.23 (2.7)	0.78 (3.0)	0.96	0.96	0.97
London	income	0.46	0.85	0.85 (0.00)	0.64 (1.0)	0.63 (1.0)	0.85	0.65	0.64
	education	0.65	0.81	0.83 (0.40)	0.74 (1.6)	0.79 (1.4)	0.86	0.77	0.79
	age	0.65	0.73	0.73 (0.17)	0.66 (1.2)	0.70 (1.7)	0.75	0.72	0.72
	election	0.67	0.73	0.81 (0.74)	0.74 (2.0)	0.76 (2.1)	0.85	0.78	0.78
Twitch	days	0.08	0.58	0.59 (0.67)	0.22 (1.4)	0.26 (1.7)	0.60	0.23	0.26

- LGC consistently outperforms SGC on all datasets (explain from graph filtering later)
- Inductive/RP always outperforms LP and the inductive base predictor

30% training; hyperparameter α , K tuned with cross-validation on training nodes

OUR MODEL CONNECTS A LOT OF DOTS

	ego features	neighboring features	neighboring labels
Label Propagation			●
OLS, MLP	●		
GNN	●	●	
GNN/RP	●	●	●

- Our generative model,

OUR MODEL CONNECTS A LOT OF DOTS

	ego features	neighboring features	neighboring labels
Label Propagation			●
OLS, MLP	●		
GNN	●	●	
GNN/RP	●	●	●

- Our generative model,
 - formalizes three types of correlations that are helpful for learning node labels

OUR MODEL CONNECTS A LOT OF DOTS

	ego features	neighboring features	neighboring labels
Label Propagation			✓
OLS, MLP	✓		
GNN	✓	✓	
GNN/RP	✓	✓	✓

- Our generative model,
 - formalizes three types of correlations that are helpful for learning node labels
 - unifies two separate lines of research

OUR MODEL CONNECTS A LOT OF DOTS

	ego features	neighboring features	neighboring labels
Label Propagation			✓
OLS, MLP	✓		
GNN	✓	✓	
GNN/RP	✓	✓	✓

- Our generative model,
 - formalizes three types of correlations that are helpful for learning node labels
 - unifies two separate lines of research
 - leads to new algorithms LGC, RP that greatly outperforms baselines

OUR MODEL CONNECTS A LOT OF DOTS

	ego features	neighboring features	neighboring labels
Label Propagation			✓
OLS, MLP	✓		
GNN	✓	✓	
GNN/RP	✓	✓	✓

- Our generative model,
 - formalizes three types of correlations that are helpful for learning node labels
 - unifies two separate lines of research
 - leads to new algorithms LGC, RP that greatly outperforms baselines
- Other important applications:

OUR MODEL CONNECTS A LOT OF DOTS

	ego features	neighboring features	neighboring labels
Label Propagation			✓
OLS, MLP	✓		
GNN	✓	✓	
GNN/RP	✓	✓	✓

- Our generative model,
 - formalizes three types of correlations that are helpful for learning node labels
 - unifies two separate lines of research
 - leads to new algorithms LGC, RP that greatly outperforms baselines
- Other important applications:
 - **How nonlinearities in GCN help?** we can sample data to test algorithms

OUR MODEL CONNECTS A LOT OF DOTS

	ego features	neighboring features	neighboring labels
Label Propagation			✓
OLS, MLP	✓		
GNN	✓	✓	
GNN/RP	✓	✓	✓

- Our generative model,
 - formalizes three types of correlations that are helpful for learning node labels
 - unifies two separate lines of research
 - leads to new algorithms LGC, RP that greatly outperforms baselines
- Other important applications:
 - How nonlinearities in GCN help? we can sample data to test algorithms
 - How are SGC and LGC different? our model helps understand smoothing

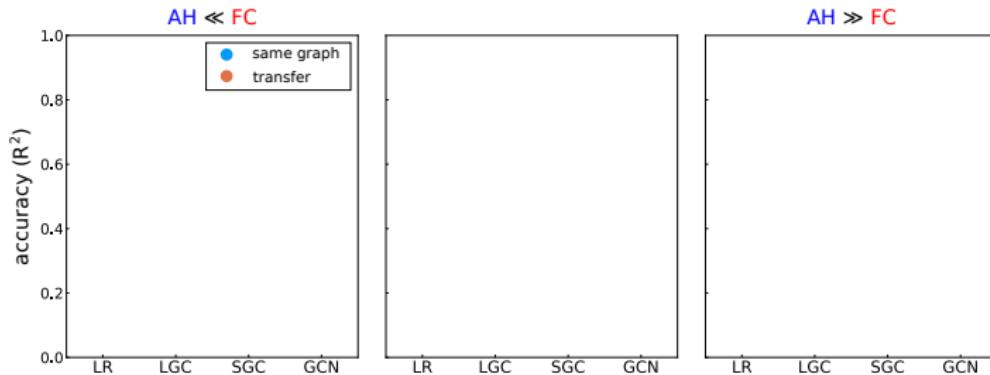
OUR MODEL CONNECTS A LOT OF DOTS

	ego features	neighboring features	neighboring labels
Label Propagation			✓
OLS, MLP	✓		
GNN	✓	✓	
GNN/RP	✓	✓	✓

- Our generative model,
 - formalizes three types of correlations that are helpful for learning node labels
 - unifies two separate lines of research
 - leads to new algorithms LGC, RP that greatly outperforms baselines
- Other important applications:
 - **How nonlinearities in GCN help?** we can sample data to test algorithms
 - **How are SGC and LGC different?** our model helps understand smoothing
 - we can use LGC to interpret empirical datasets

WE CAN SAMPLE DATA TO TEST ALGORITHMS

Transfer learning experiments on inductive methods

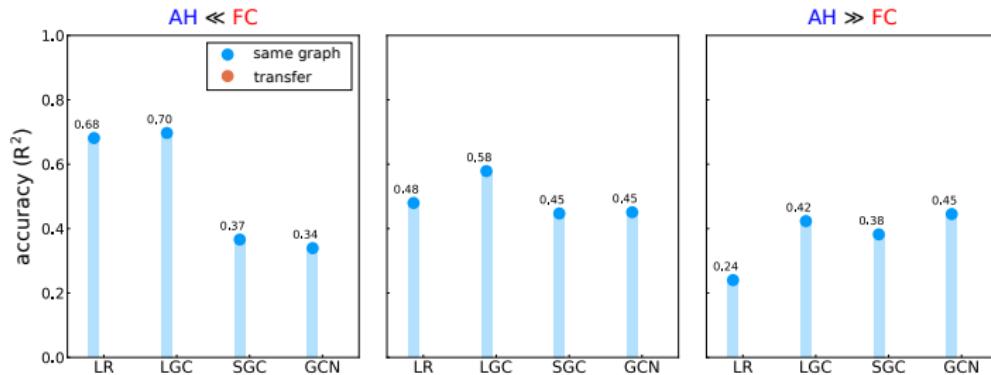


- same graph: sample one attributed graphs G_1 , train and test on different vertices

The real-world dataset we consider here is the election data from 2012/2016.

WE CAN SAMPLE DATA TO TEST ALGORITHMS

Transfer learning experiments on inductive methods

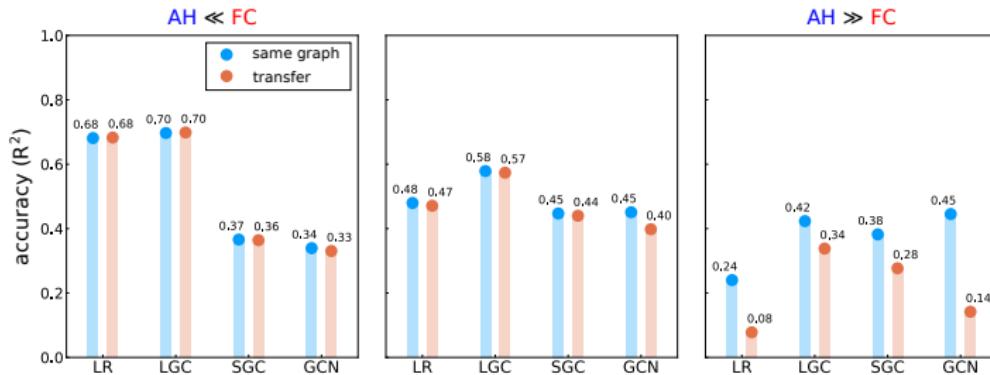


- same graph: sample one attributed graphs G_1 , train and test on different vertices
- same graph: LGC is the best when **FC** is strong, GCN is better when **AH** \gg **FC** !?!

The real-world dataset we consider here is the election data from 2012/2016.

WE CAN SAMPLE DATA TO TEST ALGORITHMS

Transfer learning experiments on inductive methods

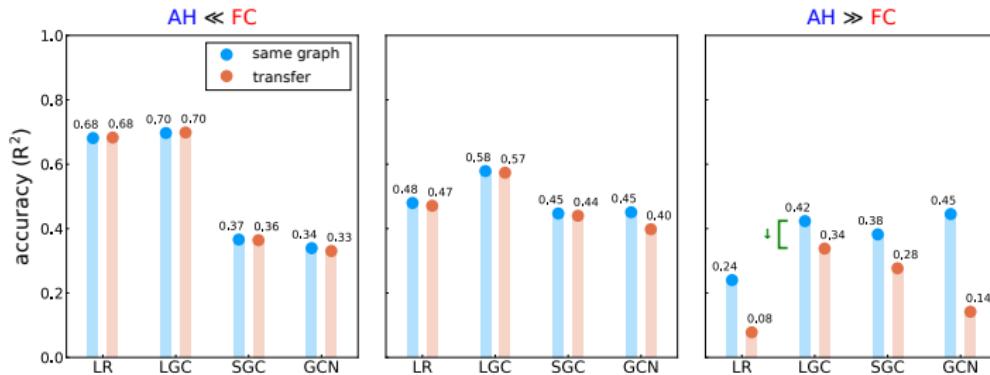


- same graph: sample one attributed graphs G_1 , train and test on different vertices
transfer: sample two attributed graphs G_1, G_2 , train on G_1 and test on G_2
- same graph: LGC is the best when **FC** is strong, GCN is better when **AH** \gg **FC** !?!

The real-world dataset we consider here is the election data from 2012/2016.

WE CAN SAMPLE DATA TO TEST ALGORITHMS

Transfer learning experiments on inductive methods

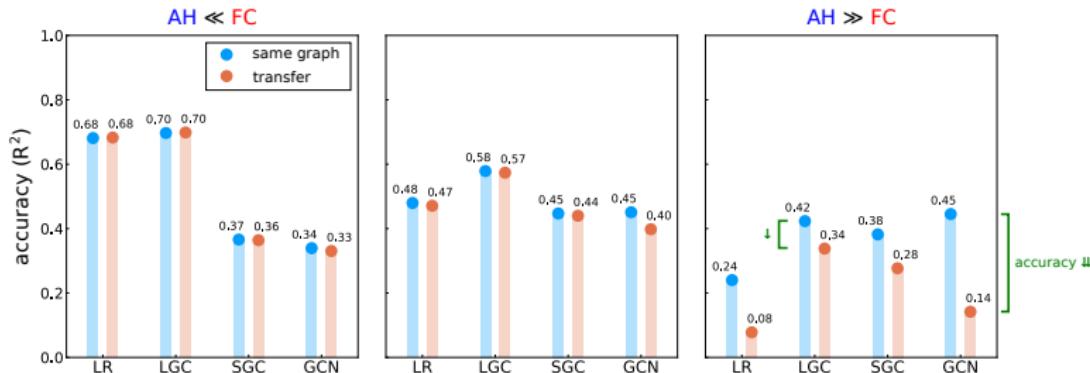


- same graph: sample one attributed graphs G_1 , train and test on different vertices
transfer: sample two attributed graphs G_1, G_2 , train on G_1 and test on G_2
- same graph: LGC is the best when **FC** is strong, GCN is better when **AH \gg FC !?**
- performance degradation happens on transfer learning when **AH \gg FC**

The real-world dataset we consider here is the election data from 2012/2016.

WE CAN SAMPLE DATA TO TEST ALGORITHMS

Transfer learning experiments on inductive methods

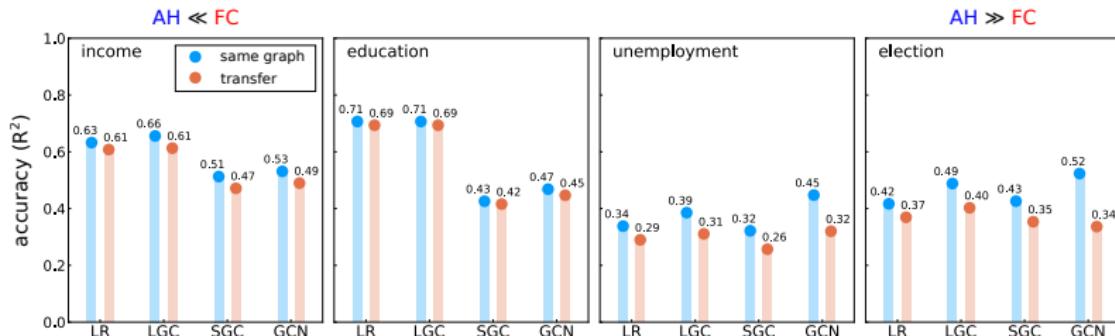


- same graph: sample one attributed graphs G_1 , train and test on different vertices
transfer: sample two attributed graphs G_1, G_2 , train on G_1 and test on G_2
- same graph: LGC is the best when **FC** is strong, GCN is better when **AH \gg FC !?**
- performance degradation happens on transfer learning when **AH \gg FC**
GCN performs much worse than LGC due to memorization

The real-world dataset we consider here is the election data from 2012/2016.

WE CAN SAMPLE DATA TO TEST ALGORITHMS

Transfer learning experiments on inductive methods

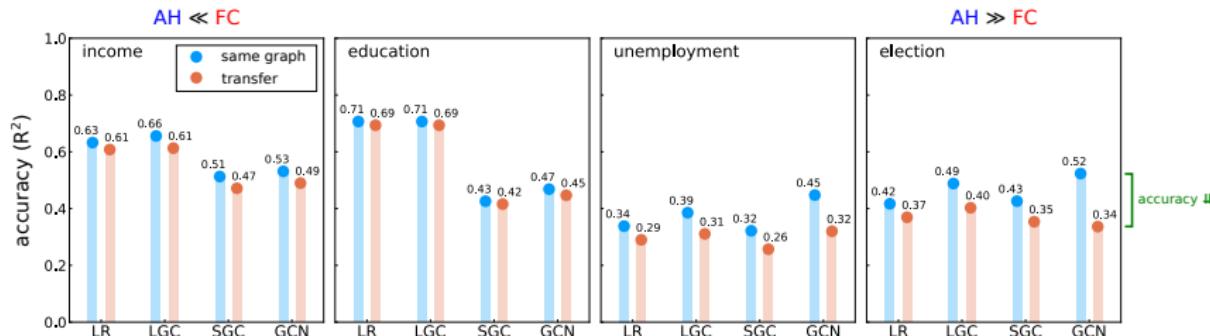


- same graph: sample one attributed graphs G_1 , train and test on different vertices
transfer: sample two attributed graphs G_1, G_2 , train on G_1 and test on G_2
- same graph: LGC is the best when **FC** is strong, GCN is better when **AH \gg FC !?!**
- performance degradation happens on transfer learning when **AH \gg FC**
GCN performs much worse than LGC due to memorization
on real-world datasets,

The real-world dataset we consider here is the election data from 2012/2016.

WE CAN SAMPLE DATA TO TEST ALGORITHMS

Transfer learning experiments on inductive methods



- same graph: sample one attributed graphs G_1 , train and test on different vertices
transfer: sample two attributed graphs G_1, G_2 , train on G_1 and test on G_2
- same graph: LGC is the best when **FC** is strong, GCN is better when **AH \gg FC !?!**
- performance degradation happens on transfer learning when **AH \gg FC**
GCN performs much worse than LGC due to memorization
on real-world datasets, we also observe similar patterns

The real-world dataset we consider here is the election data from 2012/2016.

OUR MODEL HELPS UNDERSTAND SMOOTHING



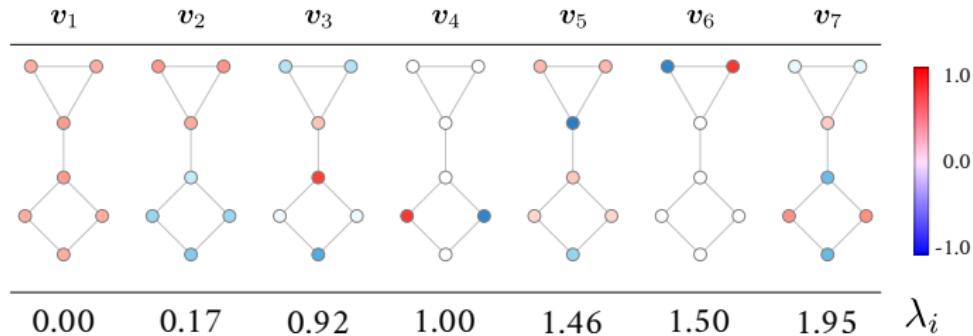
- Spectral graph theory measures smoothness with eigenvectors of $N = V\Lambda V^\top$.

OUR MODEL HELPS UNDERSTAND SMOOTHING



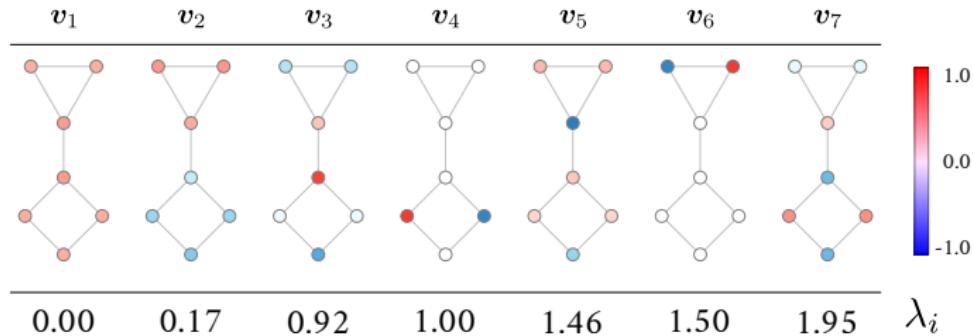
- Spectral graph theory measures smoothness with eigenvectors of $N = \mathbf{V}\Lambda\mathbf{V}^\top$.

OUR MODEL HELPS UNDERSTAND SMOOTHING



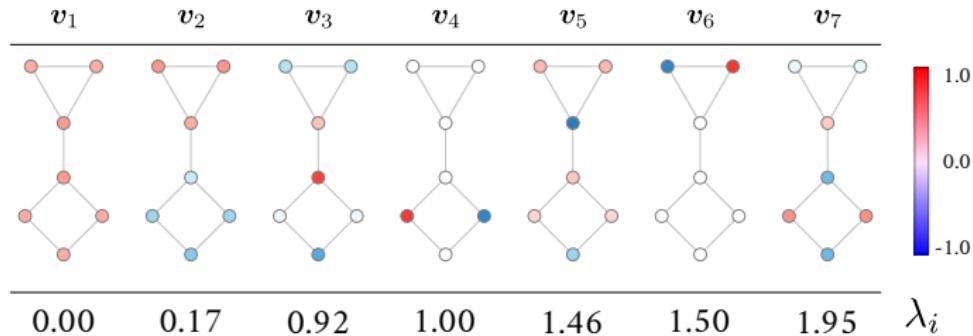
- Spectral graph theory measures smoothness with eigenvectors of $N = V\Lambda V^\top$.

OUR MODEL HELPS UNDERSTAND SMOOTHING



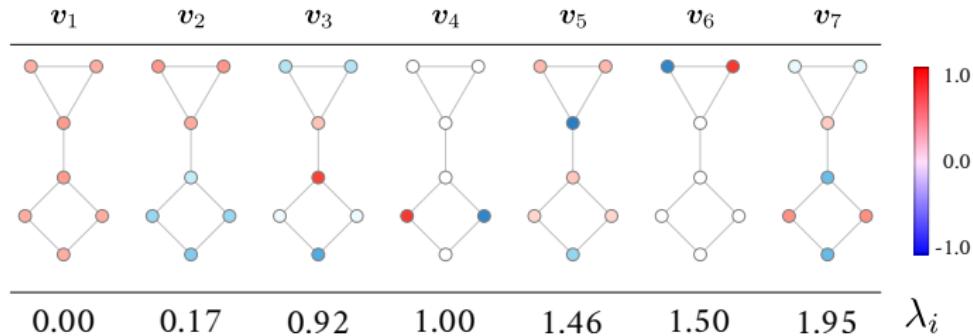
- Spectral graph theory measures smoothness with eigenvectors of $N = V\Lambda V^\top$. any graph signal can be decomposed in the eigenbasis $f = \sum_i c_i v_i$

OUR MODEL HELPS UNDERSTAND SMOOTHING



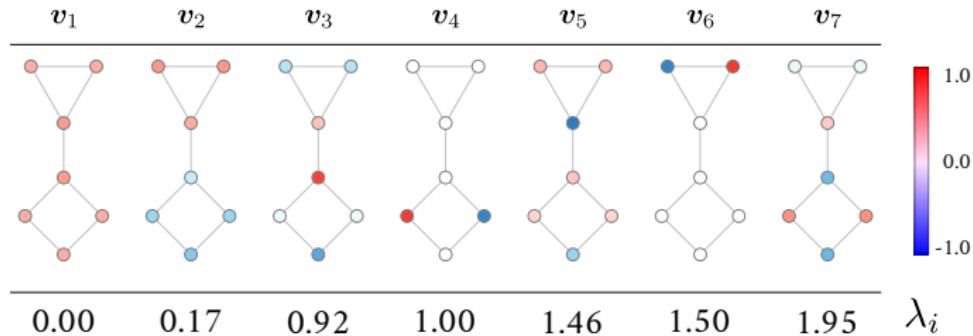
- Spectral graph theory measures smoothness with eigenvectors of $N = V\Lambda V^\top$. any graph signal can be decomposed in the eigenbasis $f = \sum_i c_i v_i$
- Why feature smoothing helps prediction? How are LGC and SGC different?

OUR MODEL HELPS UNDERSTAND SMOOTHING



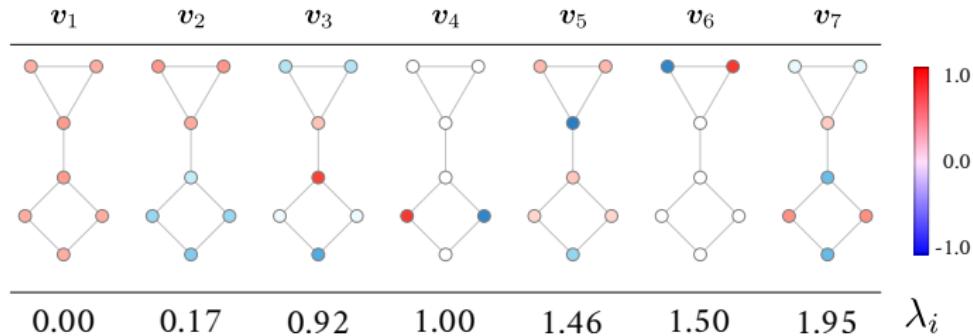
- Spectral graph theory measures smoothness with eigenvectors of $N = V\Lambda V^\top$. any graph signal can be decomposed in the eigenbasis $f = \sum_i c_i v_i$
- Why feature smoothing helps prediction? How are LGC and SGC different?
- signals tend to be smooth, noise oscillatory; reduce noise while preserving the signals

OUR MODEL HELPS UNDERSTAND SMOOTHING



- Spectral graph theory measures smoothness with eigenvectors of $N = V\Lambda V^\top$. any graph signal can be decomposed in the eigenbasis $f = \sum_i c_i v_i$
- Why feature smoothing helps prediction? How are LGC and SGC different?
- signals tend to be smooth, noise oscillatory; reduce noise while preserving the signals both LGC and SGC are low-pass filters that reduce oscillatory components

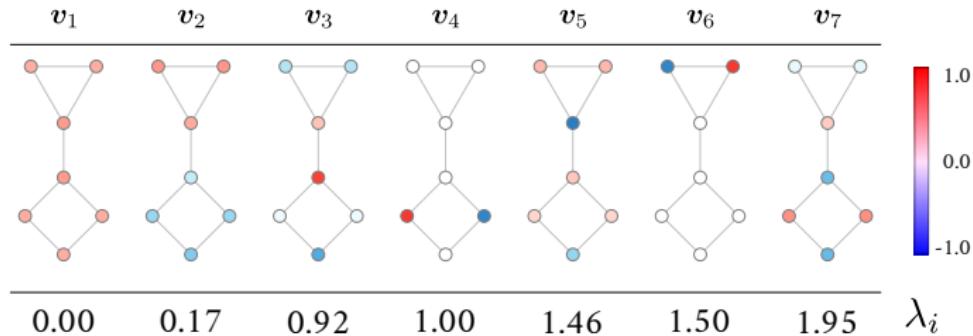
OUR MODEL HELPS UNDERSTAND SMOOTHING



- Spectral graph theory measures smoothness with eigenvectors of $N = V\Lambda V^\top$. any graph signal can be decomposed in the eigenbasis $f = \sum_i c_i v_i$
- Why feature smoothing helps prediction? How are LGC and SGC different?
- signals tend to be smooth, noise oscillatory; reduce noise while preserving the signals both LGC and SGC are low-pass filters that reduce oscillatory components

$$\text{LGC : } (I + \omega N)^{-1} f = \sum_i c_i v_i (1 + \omega \lambda_i)^{-1}$$

OUR MODEL HELPS UNDERSTAND SMOOTHING



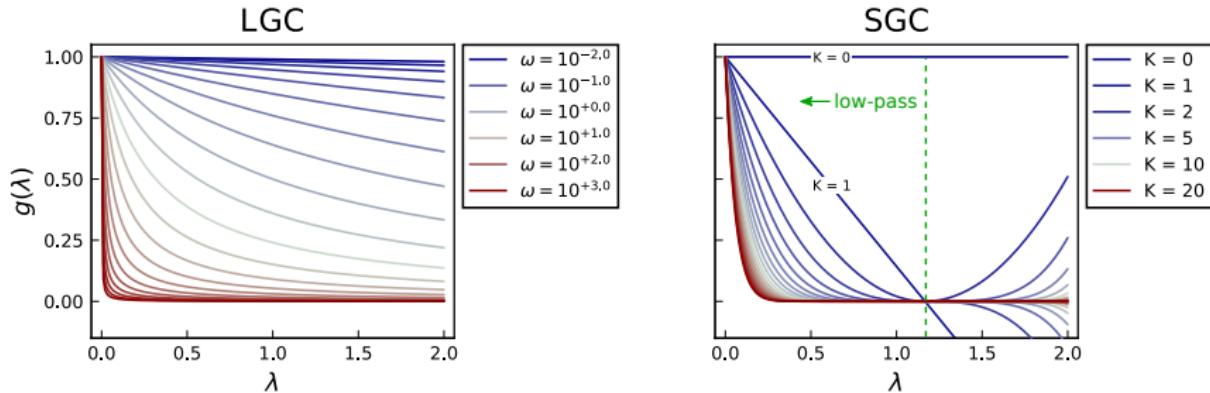
- Spectral graph theory measures smoothness with eigenvectors of $N = V\Lambda V^\top$. any graph signal can be decomposed in the eigenbasis $f = \sum_i c_i v_i$
- Why feature smoothing helps prediction? How are LGC and SGC different?
- signals tend to be smooth, noise oscillatory; reduce noise while preserving the signals both LGC and SGC are low-pass filters that reduce oscillatory components

$$\text{LGC : } (I + \omega N)^{-1} f = \sum_i c_i v_i \quad (1 + \omega \lambda_i)^{-1}$$

$$\text{SGC : } \tilde{S}^K f = \sum_i c_i v_i \quad (1 - d/(d+1)\lambda_i)^K$$

[Ortega+ 2018; Li+ 2018; Li+ 2019]

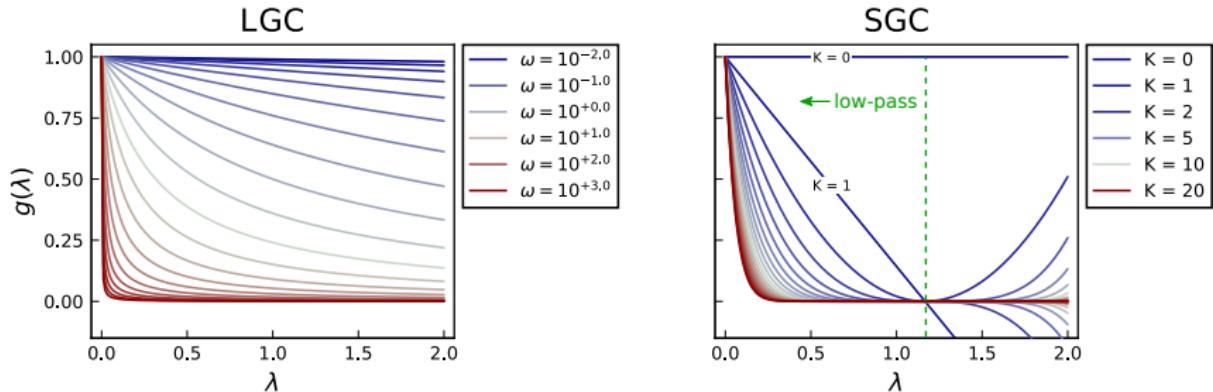
OUR MODEL HELPS UNDERSTAND SMOOTHING



$$(\mathbf{I} + \omega \mathbf{N})^{-1} \mathbf{f} = \sum_i c_i \mathbf{v}_i \ (1 + \omega \lambda_i)^{-1}$$

$$\tilde{\mathbf{S}}^K \mathbf{f} = \sum_i c_i \mathbf{v}_i \ (1 - d/(d+1)\lambda_i)^K$$

OUR MODEL HELPS UNDERSTAND SMOOTHING

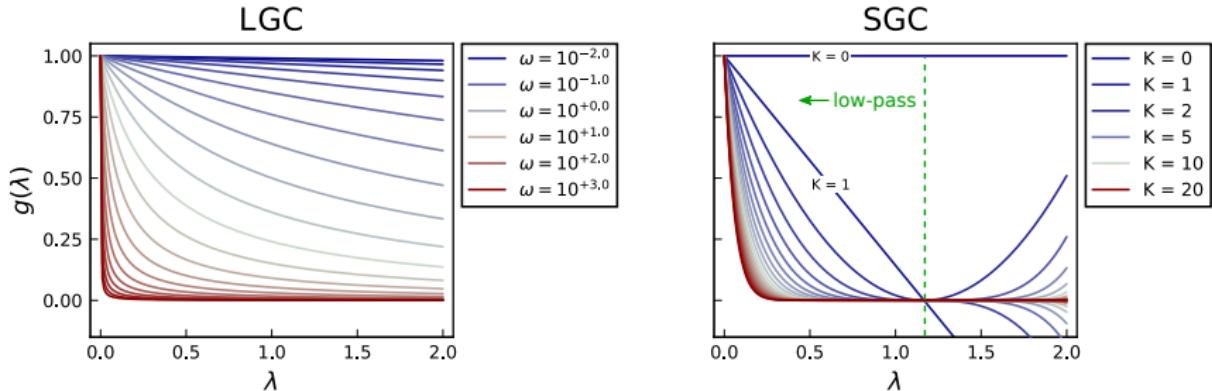


$$(\mathbf{I} + \omega \mathbf{N})^{-1} \mathbf{f} = \sum_i c_i \mathbf{v}_i \ (1 + \omega \lambda_i)^{-1}$$

$$\tilde{\mathbf{S}}^K \mathbf{f} = \sum_i c_i \mathbf{v}_i \ (1 - d/(d+1)\lambda_i)^K$$

- LGC is a low-pass filter on the entire eigenspectrum $[0, 2]$

OUR MODEL HELPS UNDERSTAND SMOOTHING

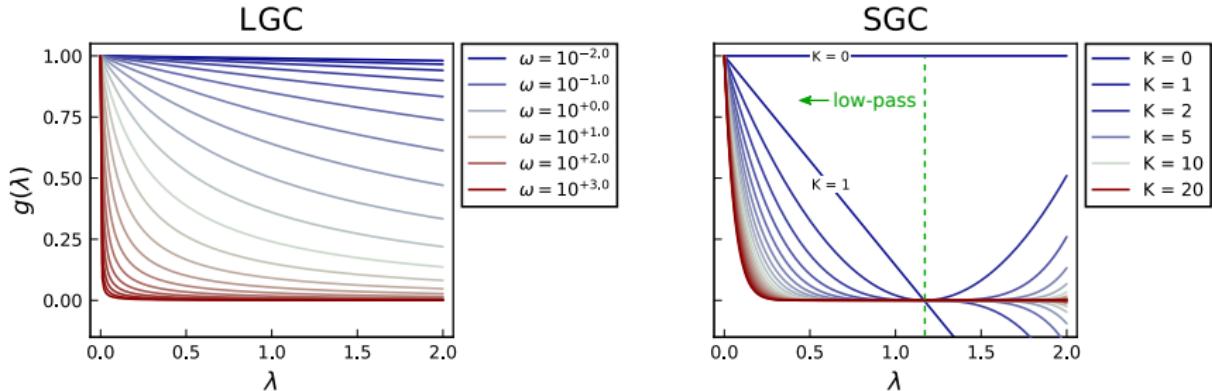


$$(\mathbf{I} + \omega \mathbf{N})^{-1} \mathbf{f} = \sum_i c_i \mathbf{v}_i \ (1 + \omega \lambda_i)^{-1}$$

$$\tilde{\mathbf{S}}^K \mathbf{f} = \sum_i c_i \mathbf{v}_i \ (1 - d/(d+1)\lambda_i)^K$$

- LGC is a low-pass filter on the entire eigenspectrum $[0, 2]$
- LGC is more flexible than SGC,

OUR MODEL HELPS UNDERSTAND SMOOTHING

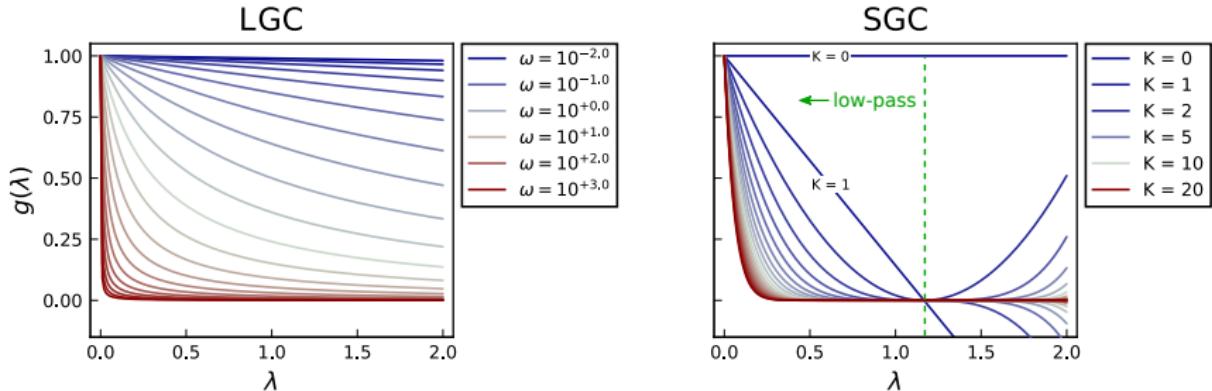


$$(\mathbf{I} + \omega \mathbf{N})^{-1} \mathbf{f} = \sum_i c_i \mathbf{v}_i \ (1 + \omega \lambda_i)^{-1}$$

$$\tilde{\mathbf{S}}^K \mathbf{f} = \sum_i c_i \mathbf{v}_i \ (1 - d/(d+1)\lambda_i)^K$$

- LGC is a low-pass filter on the entire eigenspectrum $[0, 2]$
- LGC is more flexible than SGC,
better balance between preserving the signal ($\omega, K \downarrow$) and reducing the noise ($\omega, K \uparrow$).

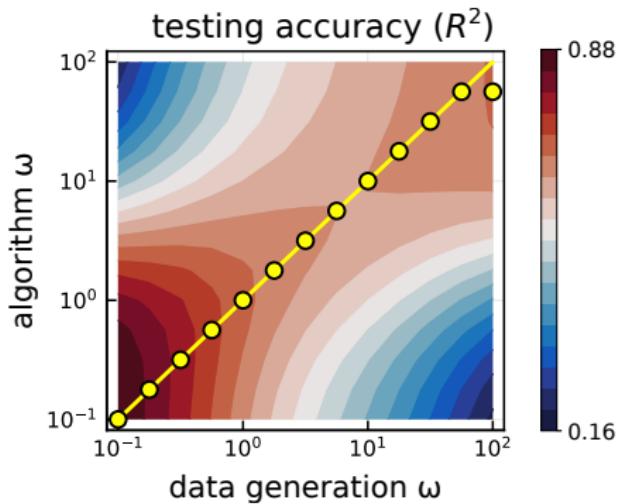
OUR MODEL HELPS UNDERSTAND SMOOTHING



$$(\mathbf{I} + \omega \mathbf{N})^{-1} \mathbf{f} = \sum_i c_i \mathbf{v}_i \ (1 + \omega \lambda_i)^{-1} \quad \tilde{\mathbf{S}}^K \mathbf{f} = \sum_i c_i \mathbf{v}_i \ (1 - d/(d+1)\lambda_i)^K$$

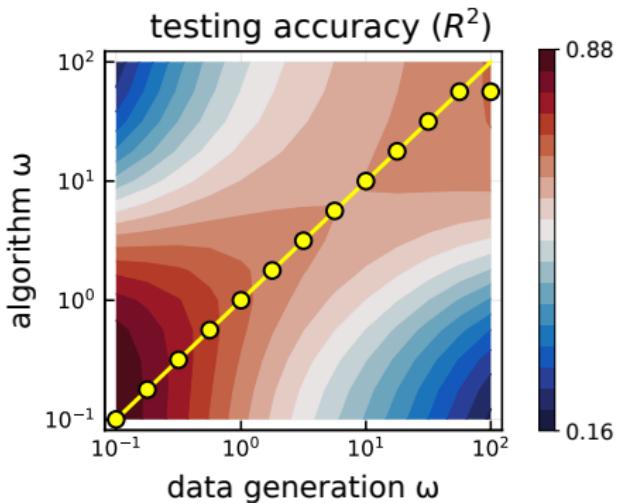
- LGC is a low-pass filter on the entire eigenspectrum $[0, 2]$
- LGC is more flexible than SGC, due to its continuous parameter ω
better balance between preserving the signal ($\omega, K \downarrow$) and reducing the noise ($\omega, K \uparrow$).

OUR MODEL HELPS UNDERSTAND SMOOTHING



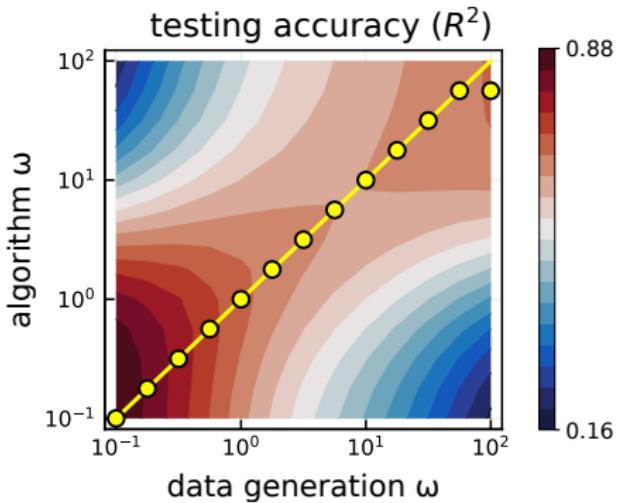
- Sample datasets from our generative model, with different $\omega = \textcolor{blue}{h}/\textcolor{red}{H}$

OUR MODEL HELPS UNDERSTAND SMOOTHING



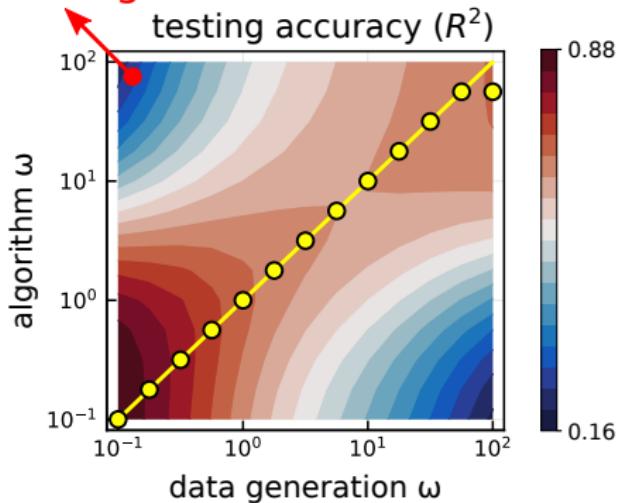
- Sample datasets from our generative model, with different $\omega = \frac{h}{H}$
For each dataset, run algo with different ω ; create a 2D performance plot

OUR MODEL HELPS UNDERSTAND SMOOTHING



- Sample datasets from our generative model, with different $\omega = h/H$
For each dataset, run algo with different ω ; create a 2D performance plot
- Performance the best along the diagonal (algo $\omega = \text{data } \omega$)

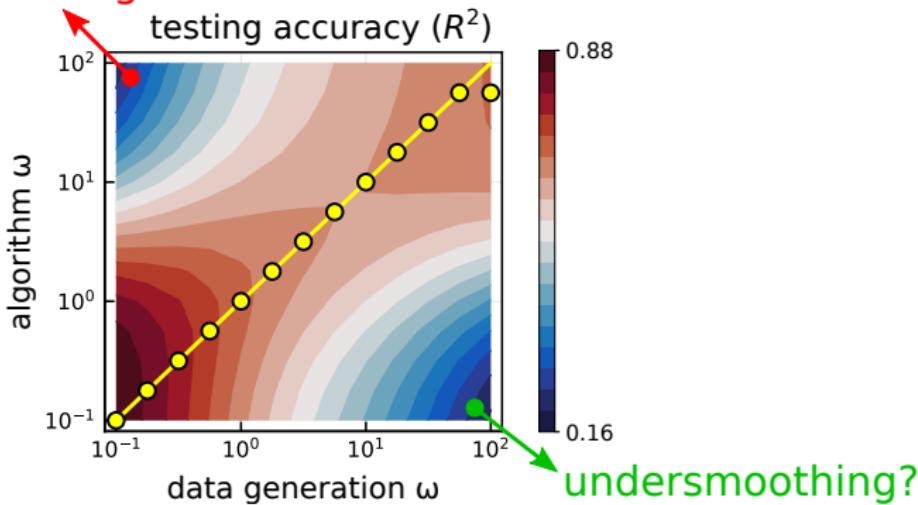
OUR MODEL HELPS UNDERSTAND SMOOTHING oversmoothing!



- Sample datasets from our generative model, with different $\omega = \frac{h}{H}$
For each dataset, run algo with different ω ; create a 2D performance plot
- Performance the best along the diagonal (algo $\omega = \text{data } \omega$)
- Oversmoothing happens if data ω is small, but algorithm ω is large

OUR MODEL HELPS UNDERSTAND SMOOTHING

oversmoothing!



- Sample datasets from our generative model, with different $\omega = h/H$
For each dataset, run algo with different ω ; create a 2D performance plot
- Performance the best along the diagonal (algo $\omega = \text{data } \omega$)
- Oversmoothing happens if data ω is small, but algorithm ω is large

WE CAN USE LGC TO INTERPRET EMPIRICAL DATA

year	sh050m	sh100m	sh500m	income	migration	birth	death	education	unemployment
2012	0.06	-0.42	0.24	0.22	0.16	-0.13	0.04	-0.90	-0.38
2016	-0.02	-0.38	0.22	0.70	0.21	-0.13	0.51	-1.53	-0.39

sh050m/sh100m/sh500m: shares of friends within 50/100/500 miles; features normalized to zero mean & unit variance

WE CAN USE LGC TO INTERPRET EMPIRICAL DATA

year	sh050m	sh100m	sh500m	income	migration	birth	death	education	unemployment
2012	0.06	-0.42	0.24	0.22	0.16	-0.13	0.04	-0.90	-0.38
2016	-0.02	-0.38	0.22	0.70	0.21	-0.13	0.51	-1.53	-0.39

- predict election margin of victory (GOP positive, DEM negiative)

sh050m/sh100m/sh500m: shares of friends within 50/100/500 miles; features normalized to zero mean & unit variance

WE CAN USE LGC TO INTERPRET EMPIRICAL DATA

year	sh050m	sh100m	sh500m	income	migration	birth	death	education	unemployment
2012	0.06	-0.42	0.24	0.22	0.16	-0.13	0.04	-0.90	-0.38
2016	-0.02	-0.38	0.22	0.70	0.21	-0.13	0.51	-1.53	-0.39

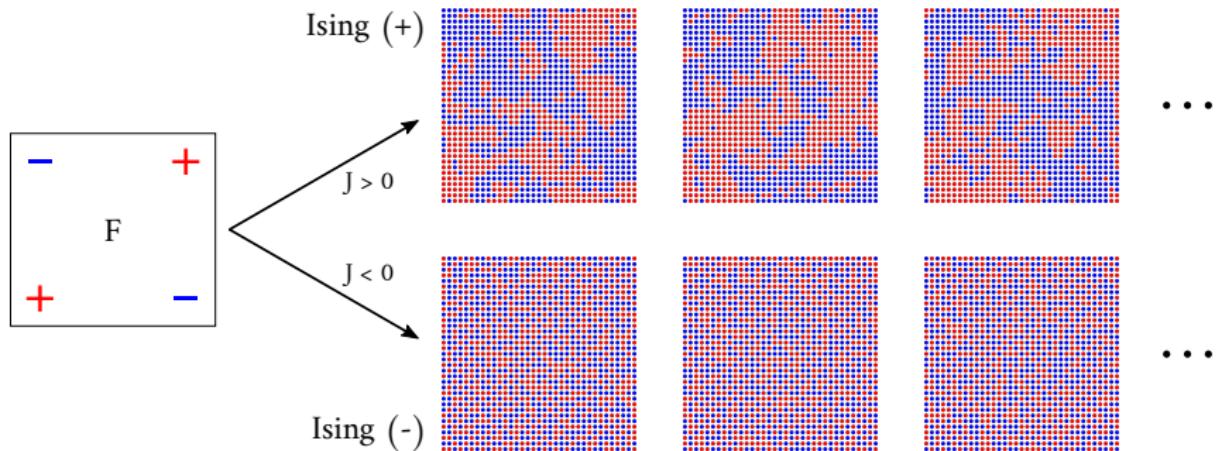
- predict election margin of victory (GOP positive, DEM negative)
- GOP leaning counties tend to have:
 - lower education levels, and higher income; this trend is stronger in 2016 than 2012
 - lower birth rate, and higher death rate → older population
 - lower unemployment rate, and higher migration rate → manufacturing hubs?
 - lower sh100m, and higher sh500m → rural areas?

sh050m/sh100m/sh500m: shares of friends within 50/100/500 miles; features normalized to zero mean & unit variance

WHERE DO WE GO FROM HERE?

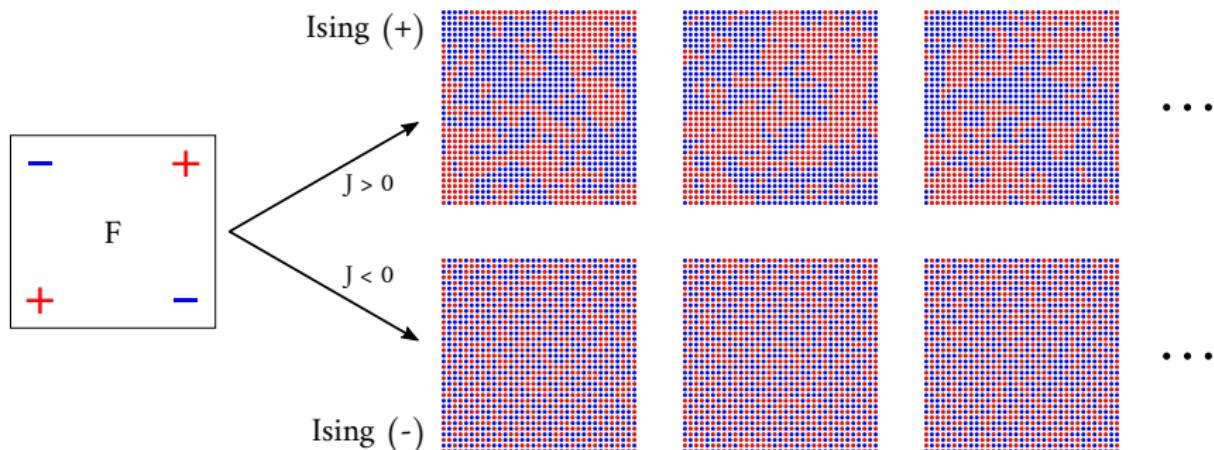
- What if the labels are not positively correlated?
(e.g. heterophily graphs)
- What if the labels are categorical variables?
(node classification setting)

WHAT IF LABELS AREN'T POSITIVELY CORRELATED?



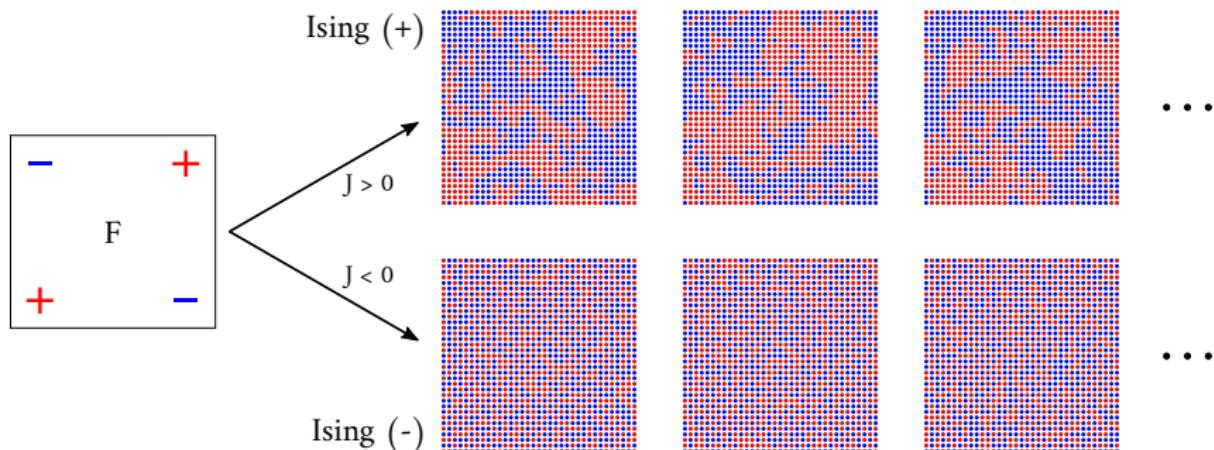
- Sometimes labels can be negatively correlated among neighbors.

WHAT IF LABELS AREN'T POSITIVELY CORRELATED?



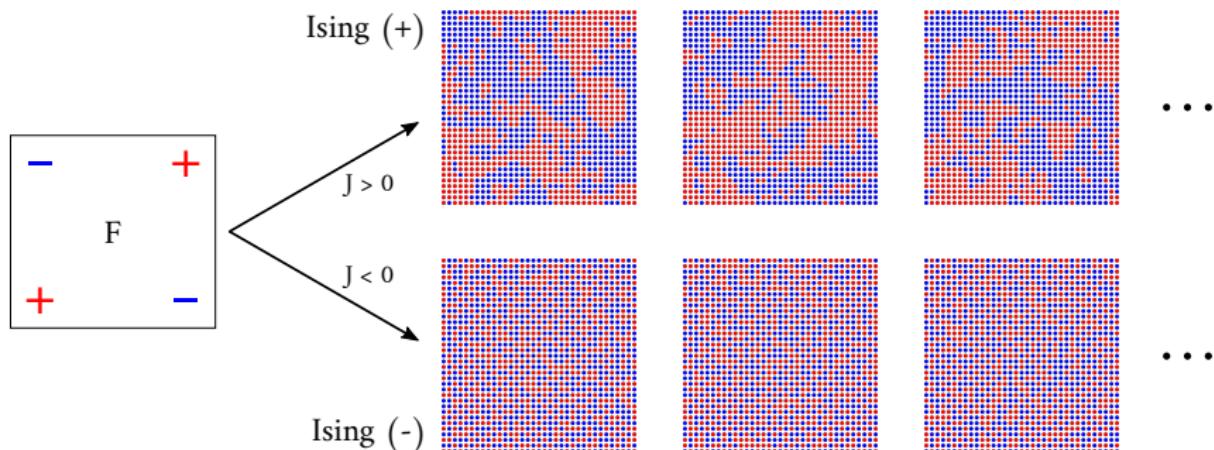
- Sometimes labels can be negatively correlated among neighbors.
 - Ising model with negative coupling J (antiferromagnetic materials)

WHAT IF LABELS AREN'T POSITIVELY CORRELATED?



- Sometimes labels can be negatively correlated among neighbors.
 - Ising model with negative coupling J (antiferromagnetic materials)
 - gender in sexual interaction network

WHAT IF LABELS AREN'T POSITIVELY CORRELATED?



- Sometimes labels can be negatively correlated among neighbors.
 - Ising model with negative coupling J (antiferromagnetic materials)
 - gender in sexual interaction network
- Residuals are negatively correlated among neighbors as well.

WE CAN LEARN THE CORRELATION DIRECTLY

- Model the residual with a multivariate Gaussian:

$$\mathbf{r} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Gamma}^{-1}), \quad \boldsymbol{\Gamma} = \beta(\mathbf{I} - \alpha \mathbf{S}), \quad \mathbf{S} = \mathbf{D}^{-1/2} \mathbf{W} \mathbf{D}^{-1/2}$$

WE CAN LEARN THE CORRELATION DIRECTLY

- Model the residual with a multivariate Gaussian:

$$\mathbf{r} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Gamma}^{-1}), \quad \boldsymbol{\Gamma} = \beta(\mathbf{I} - \alpha \mathbf{S}), \quad \mathbf{S} = \mathbf{D}^{-1/2} \mathbf{W} \mathbf{D}^{-1/2}$$

- $\beta > 0$ controls the magnitude of the residuals

WE CAN LEARN THE CORRELATION DIRECTLY

- Model the residual with a multivariate Gaussian:

$$\mathbf{r} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Gamma}^{-1}), \quad \boldsymbol{\Gamma} = \beta(\mathbf{I} - \alpha \mathbf{S}), \quad \mathbf{S} = \mathbf{D}^{-1/2} \mathbf{W} \mathbf{D}^{-1/2}$$

- $\beta > 0$ controls the magnitude of the residuals
- $\alpha = 0$, uncorrelated residuals; i.i.d. assumption and sum-of-square loss
- $\alpha > 0$, positively correlated residuals, correct GNN with residual propagation
- $\alpha < 0$, negatively correlated residuals

WE CAN LEARN THE CORRELATION DIRECTLY

- Model the residual with a multivariate Gaussian:

$$\mathbf{r} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Gamma}^{-1}), \quad \boldsymbol{\Gamma} = \beta(\mathbf{I} - \alpha \mathbf{S}), \quad \mathbf{S} = \mathbf{D}^{-1/2} \mathbf{W} \mathbf{D}^{-1/2}$$

- $\beta > 0$ controls the magnitude of the residuals
 - $\alpha = 0$, uncorrelated residuals; i.i.d. assumption and sum-of-square loss
 - $\alpha > 0$, positively correlated residuals, correct GNN with residual propagation
 - $\alpha < 0$, negatively correlated residuals
- Testing residuals can be estimated with conditional expectation

$$\hat{\mathbf{r}}_U = E[\mathbf{r}_U | \mathbf{r}_L = \mathbf{r}_L] = -\boldsymbol{\Gamma}_{UU}^{-1} \boldsymbol{\Gamma}_{UL} \mathbf{r}_L = -(\mathbf{I} - \alpha \mathbf{S})_{UU}^{-1} (\mathbf{I} - \alpha \mathbf{S})_{UL} \mathbf{r}_L$$

WE CAN LEARN THE CORRELATION DIRECTLY

- Model the residual with a multivariate Gaussian:

$$\mathbf{r} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Gamma}^{-1}), \quad \boldsymbol{\Gamma} = \beta(\mathbf{I} - \alpha \mathbf{S}), \quad \mathbf{S} = \mathbf{D}^{-1/2} \mathbf{W} \mathbf{D}^{-1/2}$$

- $\beta > 0$ controls the magnitude of the residuals
 - $\alpha = 0$, uncorrelated residuals; i.i.d. assumption and sum-of-square loss
 - $\alpha > 0$, positively correlated residuals, correct GNN with residual propagation
 - $\alpha < 0$, negatively correlated residuals
-
- Testing residuals can be estimated with conditional expectation
- $$\hat{\mathbf{r}}_U = E[\mathbf{r}_U | \mathbf{r}_L = \mathbf{r}_L] = -\boldsymbol{\Gamma}_{UU}^{-1} \boldsymbol{\Gamma}_{UL} \mathbf{r}_L = -(\mathbf{I} - \alpha \mathbf{S})_{UU}^{-1} (\mathbf{I} - \alpha \mathbf{S})_{UL} \mathbf{r}_L$$
-
- How do we determine the parameter α ?

WE CAN LEARN THE CORRELATION DIRECTLY

- Model the residual with a multivariate Gaussian:

$$\mathbf{r} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Gamma}^{-1}), \quad \boldsymbol{\Gamma} = \beta(\mathbf{I} - \alpha \mathbf{S}), \quad \mathbf{S} = \mathbf{D}^{-1/2} \mathbf{W} \mathbf{D}^{-1/2}$$

- $\beta > 0$ controls the magnitude of the residuals
 - $\alpha = 0$, uncorrelated residuals; i.i.d. assumption and sum-of-square loss
 - $\alpha > 0$, positively correlated residuals, correct GNN with residual propagation
 - $\alpha < 0$, negatively correlated residuals
-
- Testing residuals can be estimated with conditional expectation
 - How do we determine the parameter α ?
 - learn the optimal α, β with **maximum marginal likelihood**

WE CAN LEARN THE CORRELATION DIRECTLY

- Learn model by maximizing marginal likelihood of training residuals r_L

WE CAN LEARN THE CORRELATION DIRECTLY

- Learn model by maximizing marginal likelihood of training residuals \mathbf{r}_L

$$\alpha^*, \beta^* = \arg \max_{\alpha, \beta} [\log |\boldsymbol{\Gamma}| - \log |\boldsymbol{\Gamma}_{UU}| - \mathbf{r}_L^\top (\boldsymbol{\Gamma}_{LL} - \boldsymbol{\Gamma}_{LU} \boldsymbol{\Gamma}_{UU}^{-1} \boldsymbol{\Gamma}_{UL}) \mathbf{r}_L]$$

WE CAN LEARN THE CORRELATION DIRECTLY

- Learn model by maximizing marginal likelihood of training residuals \mathbf{r}_L

$$\alpha^*, \beta^* = \arg \max_{\alpha, \beta} [\underbrace{\log |\boldsymbol{\Gamma}| - \log |\boldsymbol{\Gamma}_{UU}|}_{\text{naively cost } \mathcal{O}(|V|^3)} - \mathbf{r}_L^\top (\boldsymbol{\Gamma}_{LL} - \boldsymbol{\Gamma}_{LU} \boldsymbol{\Gamma}_{UU}^{-1} \boldsymbol{\Gamma}_{UL}) \mathbf{r}_L]$$

WE CAN LEARN THE CORRELATION DIRECTLY

- Learn model by maximizing marginal likelihood of training residuals \mathbf{r}_L

$$\alpha^*, \beta^* = \arg \max_{\alpha, \beta} \underbrace{[\log |\boldsymbol{\Gamma}| - \log |\boldsymbol{\Gamma}_{UU}| - \mathbf{r}_L^\top (\boldsymbol{\Gamma}_{LL} - \boldsymbol{\Gamma}_{LU} \boldsymbol{\Gamma}_{UU}^{-1} \boldsymbol{\Gamma}_{UL}) \mathbf{r}_L]}_{\text{naively cost } \mathcal{O}(|V|^3)}$$

- Estimate $\log |\boldsymbol{\Gamma}|$ with **stochastic trace estimator** & **Lanczos quadrature**

$$\begin{aligned} \log |\boldsymbol{\Gamma}| = \text{tr}(\log \boldsymbol{\Gamma}) &\approx \frac{1}{T} \sum_{t=1}^T \mathbf{z}_t^\top (\log \boldsymbol{\Gamma}) \mathbf{z}_t = \frac{1}{T} \sum_{t=1}^T \sum_{i=1}^n \mu_{ti}^2 \log \lambda_i(\boldsymbol{\Gamma}) \\ &\approx \frac{1}{T} \sum_{t=1}^T \sum_{i=1}^k \omega_{ti}^2 \log \xi_{ti} \end{aligned}$$

- \mathbf{z}_t random Gaussian trial vectors with unit-variance
- ω_{ti}/ξ_{ti} numerical integration weights/nodes computed with Lanczos algorithm

WE CAN LEARN THE CORRELATION DIRECTLY

- Learn model by maximizing marginal likelihood of training residuals \mathbf{r}_L

$$\alpha^*, \beta^* = \arg \max_{\alpha, \beta} \underbrace{[\log |\boldsymbol{\Gamma}| - \log |\boldsymbol{\Gamma}_{UU}| - \mathbf{r}_L^\top (\boldsymbol{\Gamma}_{LL} - \boldsymbol{\Gamma}_{LU}\boldsymbol{\Gamma}_{UU}^{-1}\boldsymbol{\Gamma}_{UL})\mathbf{r}_L]}_{\text{naively cost } \mathcal{O}(|V|^3)}$$

- Estimate $\log |\boldsymbol{\Gamma}|$ with stochastic trace estimator & Lanczos quadrature

$$\begin{aligned} \log |\boldsymbol{\Gamma}| = \text{tr}(\log \boldsymbol{\Gamma}) &\approx \frac{1}{T} \sum_{t=1}^T \mathbf{z}_t^\top (\log \boldsymbol{\Gamma}) \mathbf{z}_t = \frac{1}{T} \sum_{t=1}^T \sum_{i=1}^n \mu_{ti}^2 \log \lambda_i(\boldsymbol{\Gamma}) \\ &\approx \frac{1}{T} \sum_{t=1}^T \sum_{i=1}^k \omega_{ti}^2 \log \xi_{ti} \end{aligned}$$

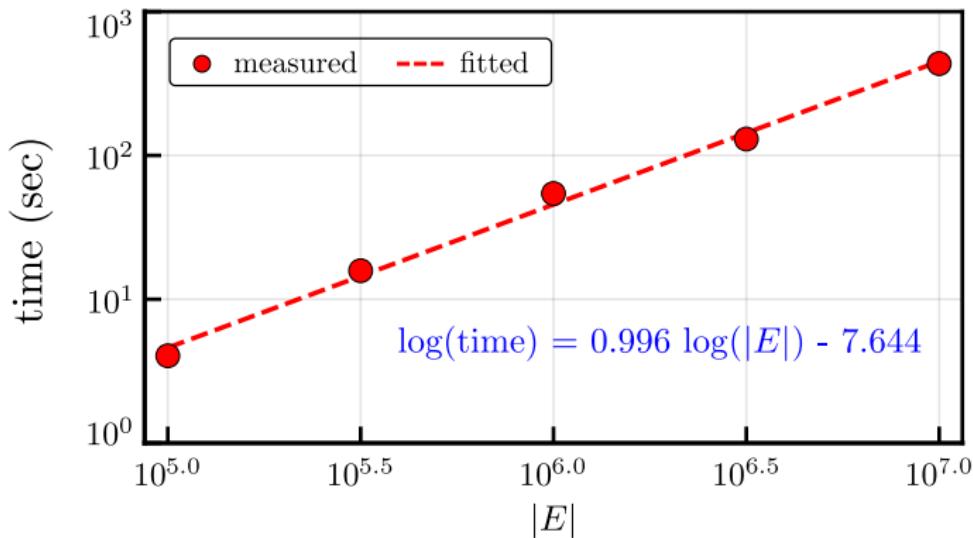
- \mathbf{z}_t random Gaussian trial vectors with unit-variance
- ω_{ti}/ξ_{ti} numerical integration weights/nodes computed with Lanczos algorithm

- Estimate gradient with stochastic trace estimator & CG solver

$$\frac{\partial \log |\boldsymbol{\Gamma}|}{\partial \alpha} \approx \frac{1}{T} \sum_{t=1}^T (\boldsymbol{\Gamma}^{-1} \mathbf{z}_t)^\top \left(\frac{\partial \boldsymbol{\Gamma}}{\partial \alpha} \mathbf{z}_t \right)$$

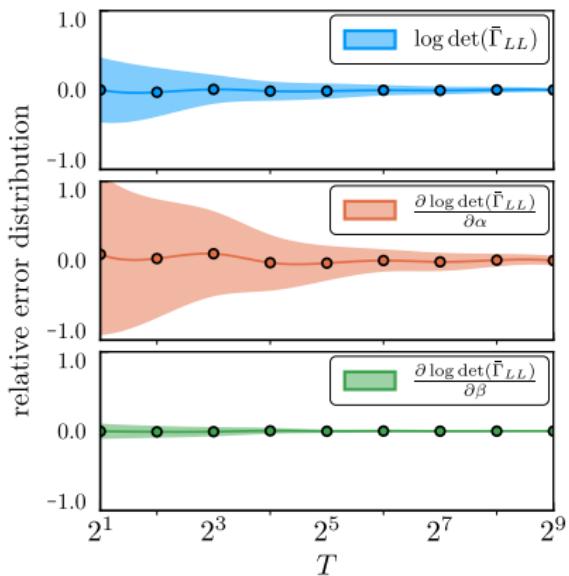
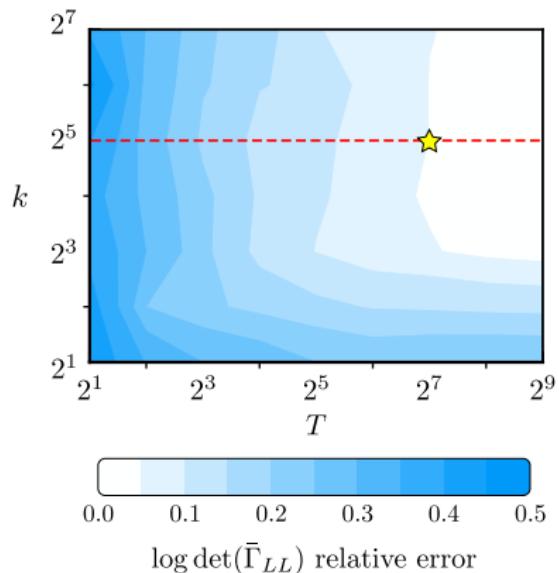
[Ubaru+ 2017; Gardner+ 2018]

STOCHASTIC ESTIMATION IS EFFICIENT



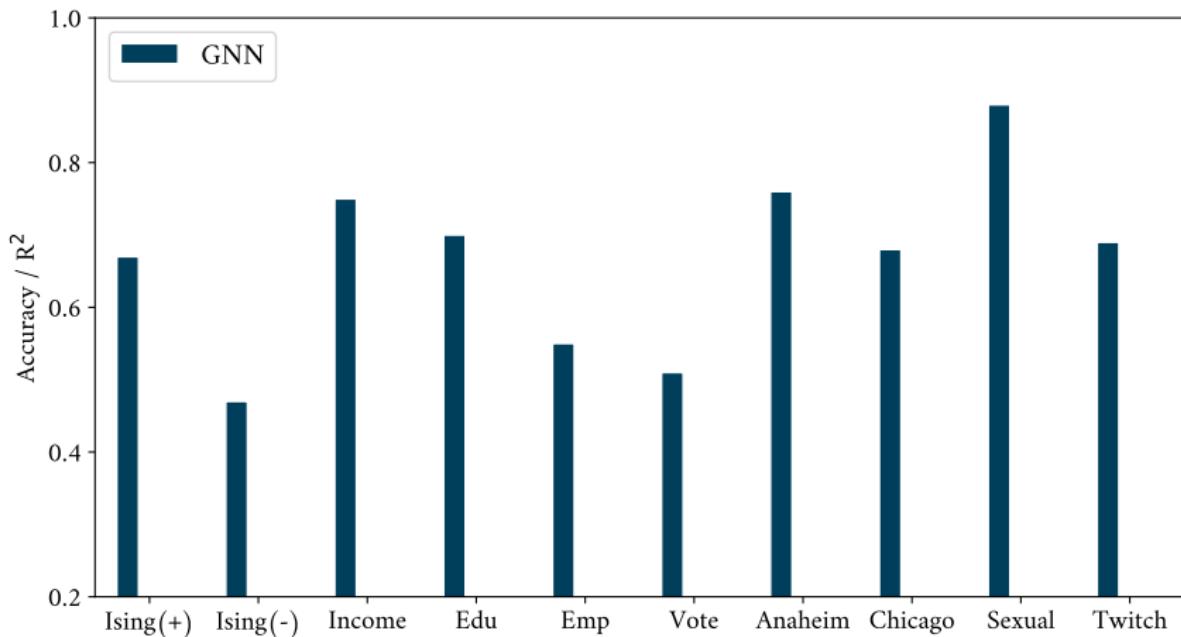
- Stochastic trace estimator cost $\mathcal{O}(|E|)$, linear in the # of edges

STOCHASTIC ESTIMATION IS ACCURATE

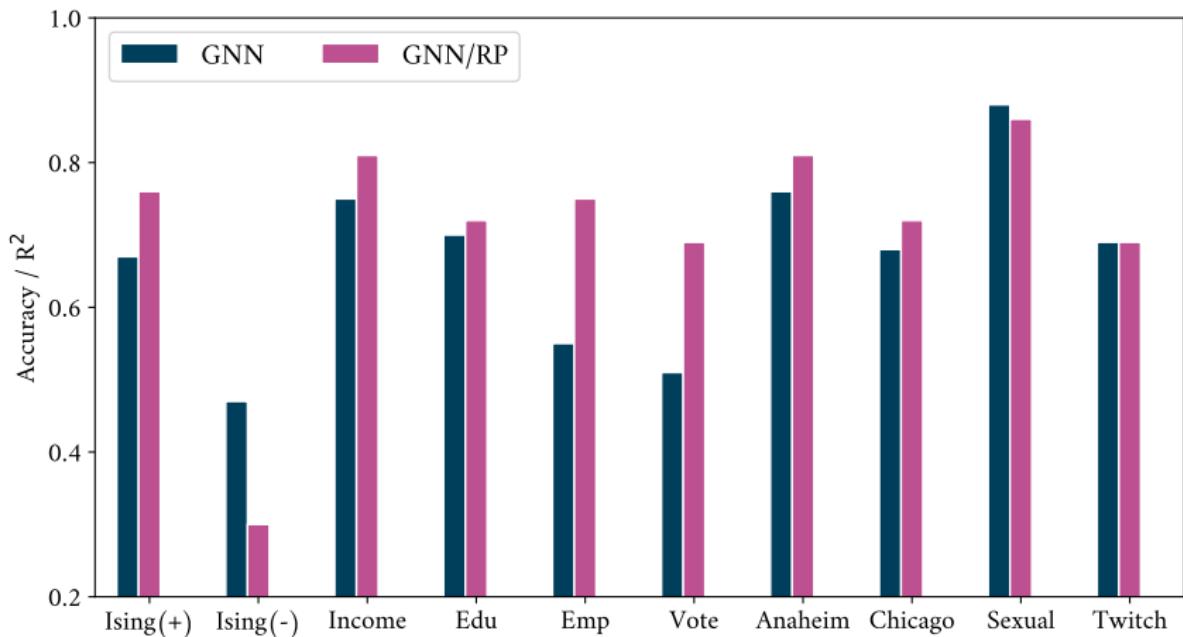


- Stochastic trace estimator gives accurate results
 - a small number of CG iterations, Gaussian trial vectors; $< 5\%$ error
 - unbiased estimation of gradient optimize with SGD

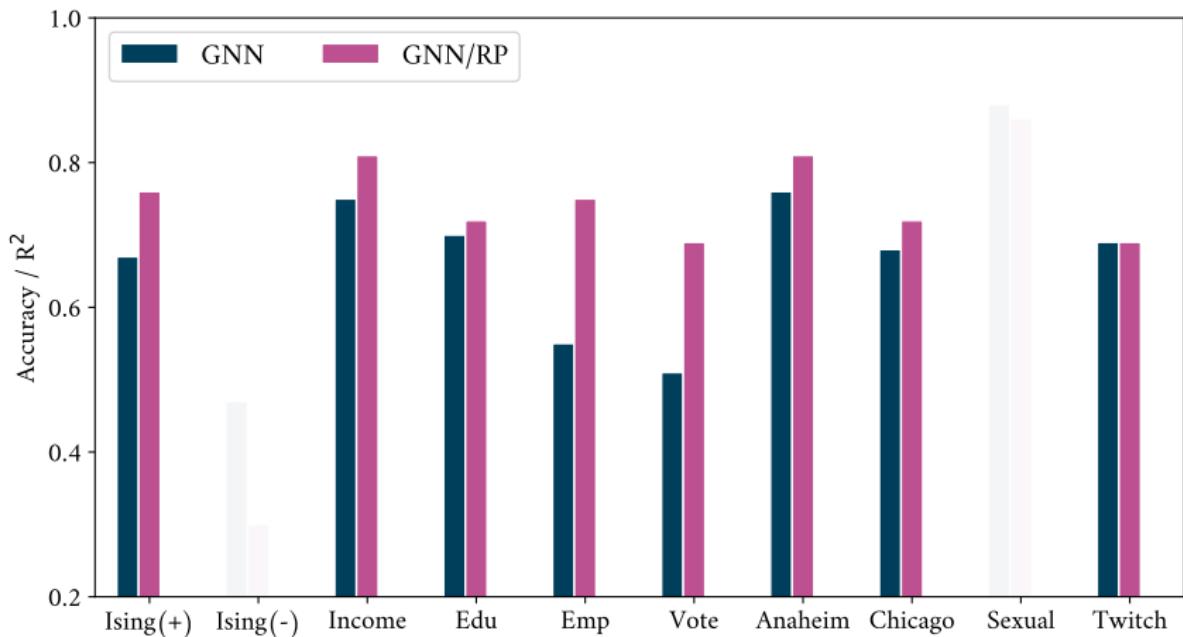
C-GNN PERFORMS WELL ON HETEROGRAPHY GRAPHS



C-GNN PERFORMS WELL ON HETEROGRAPHY GRAPHS

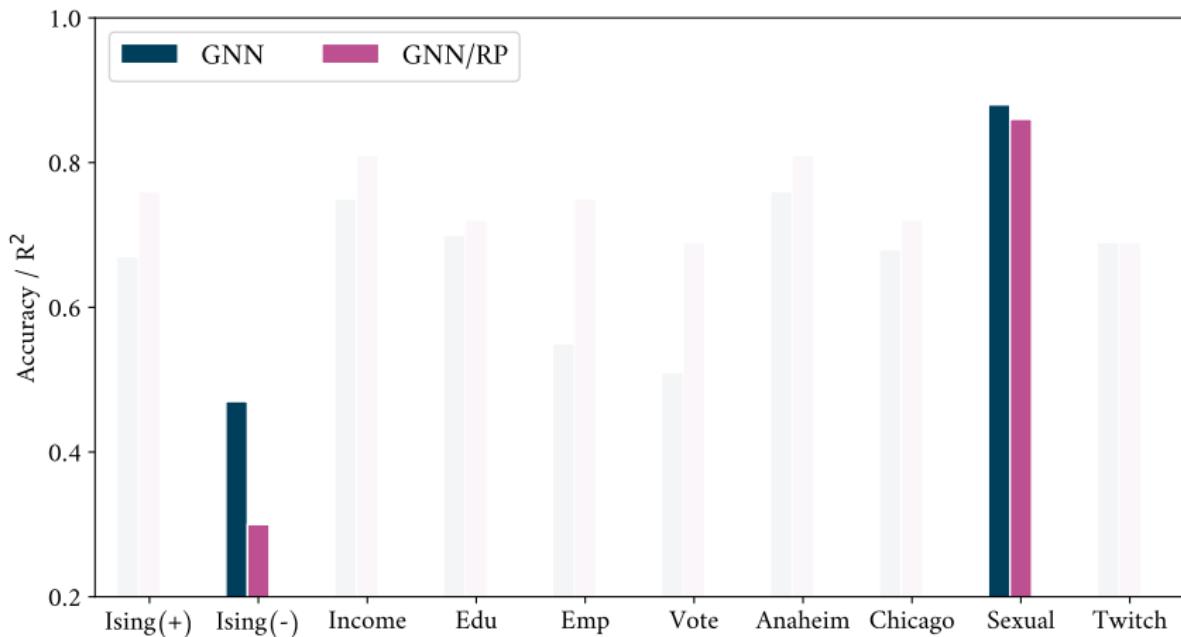


C-GNN PERFORMS WELL ON HETEROGRAPHY GRAPHS



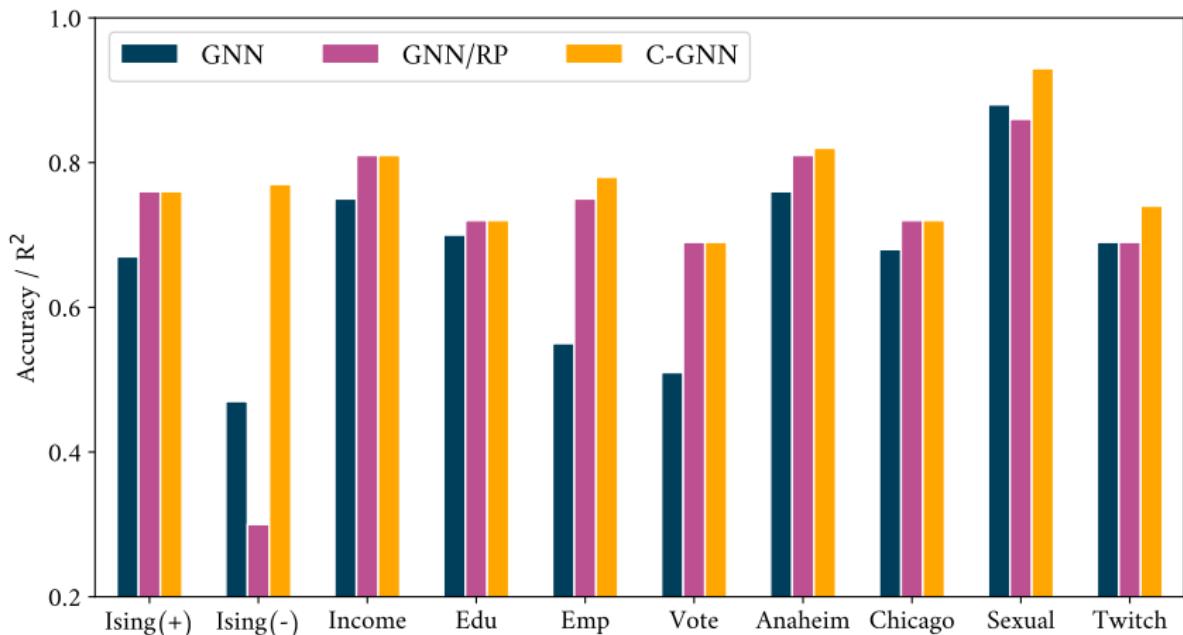
- GNN/RP outperforms GNN on homophily graphs;

C-GNN PERFORMS WELL ON HETEROGRAPHY GRAPHS



- GNN/RP outperforms GNN on homophily graphs; underperforms if heterophily

C-GNN PERFORMS WELL ON HETEROGRAPHY GRAPHS



- GNN/RP outperforms GNN on homophily graphs; underperforms if heterophily
- C-GNN performs the best on all datasets

WHERE DO WE GO FROM HERE?

- What if the labels are not positively correlated?
(e.g. heterophily graphs)
- What if the labels are categorical variables?
(node classification setting)

A STATISTICAL MODEL FOR ATTRIBUTED GRAPHS

- Consider a two-step process for generating node attributes:

$\mathbf{h} \in \mathbb{R}^c$, $\mathbf{H} \in \mathbb{R}^{c \times c}$ are model parameters with all positive entries

A STATISTICAL MODEL FOR ATTRIBUTED GRAPHS

- Consider a two-step process for generating node attributes:
 - Sample node labels jointly from a pairwise MRF

$$P(\mathbf{y}) = \frac{\varphi(\mathbf{y})}{\sum_{\mathbf{y}'} \varphi(\mathbf{y}')}, \quad \varphi(\mathbf{y}) = \prod_{u \in V} h(y_u) \prod_{(u,v) \in E} H(y_u, y_v)$$

$\mathbf{h} \in \mathbb{R}^c$, $\mathbf{H} \in \mathbb{R}^{c \times c}$ are model parameters with all positive entries

A STATISTICAL MODEL FOR ATTRIBUTED GRAPHS

- Consider a two-step process for generating node attributes:
 - Sample node labels jointly from a pairwise MRF

$$P(\mathbf{y}) = \frac{\varphi(\mathbf{y})}{\sum_{\mathbf{y}'} \varphi(\mathbf{y}')}, \quad \varphi(\mathbf{y}) = \prod_{u \in V} h(y_u) \prod_{(u,v) \in E} H(y_u, y_v)$$

- Given the node labels, sample the features from $P(\mathbf{X}|\mathbf{y})$

$$P(\mathbf{X}|\mathbf{y}) = \prod_{u \in V} P(\mathbf{x}_u|y_u)$$

$\mathbf{h} \in \mathbb{R}^c$, $\mathbf{H} \in \mathbb{R}^{c \times c}$ are model parameters with all positive entries

A STATISTICAL MODEL FOR ATTRIBUTED GRAPHS

- Consider a two-step process for generating node attributes:
 - Sample node labels jointly from a pairwise MRF

$$P(\mathbf{y}) = \frac{\varphi(\mathbf{y})}{\sum_{\mathbf{y}'} \varphi(\mathbf{y}')}, \quad \varphi(\mathbf{y}) = \prod_{u \in V} h(y_u) \prod_{(u,v) \in E} H(y_u, y_v)$$

- Given the node labels, sample the features from $P(\mathbf{X}|\mathbf{y})$

$$P(\mathbf{X}|\mathbf{y}) = \prod_{u \in V} P(\mathbf{x}_u|y_u)$$

- Given the observation \mathbf{X} , the posterior probability is,

$\mathbf{h} \in \mathbb{R}^c$, $\mathbf{H} \in \mathbb{R}^{c \times c}$ are model parameters with all positive entries

A STATISTICAL MODEL FOR ATTRIBUTED GRAPHS

- Consider a two-step process for generating node attributes:
 - Sample node labels jointly from a pairwise MRF

$$P(\mathbf{y}) = \frac{\varphi(\mathbf{y})}{\sum_{\mathbf{y}'} \varphi(\mathbf{y}')}, \quad \varphi(\mathbf{y}) = \prod_{u \in V} h(y_u) \prod_{(u,v) \in E} H(y_u, y_v)$$

- Given the node labels, sample the features from $P(\mathbf{X}|\mathbf{y})$

$$P(\mathbf{X}|\mathbf{y}) = \prod_{u \in V} P(\mathbf{x}_u|y_u)$$

- Given the observation \mathbf{X} , the posterior probability is,

$$P(\mathbf{y}|\mathbf{X}) = \frac{P(\mathbf{y})P(\mathbf{X}|\mathbf{y})}{\sum_{\mathbf{y}'} P(\mathbf{y}')P(\mathbf{X}|\mathbf{y}')} \cong \prod_{u \in V} P(\mathbf{x}_u|y_u) \cdot h(y_u) \prod_{(u,v) \in E} H(y_u, y_v)$$

$\mathbf{h} \in \mathbb{R}^c$, $\mathbf{H} \in \mathbb{R}^{c \times c}$ are model parameters with all positive entries

A STATISTICAL MODEL FOR ATTRIBUTED GRAPHS

- Consider a two-step process for generating node attributes:
 - Sample node labels jointly from a pairwise MRF

$$P(\mathbf{y}) = \frac{\varphi(\mathbf{y})}{\sum_{\mathbf{y}'} \varphi(\mathbf{y}')}, \quad \varphi(\mathbf{y}) = \prod_{u \in V} h(y_u) \prod_{(u,v) \in E} H(y_u, y_v)$$

- Given the node labels, sample the features from $P(\mathbf{X}|\mathbf{y})$

$$P(\mathbf{X}|\mathbf{y}) = \prod_{u \in V} P(\mathbf{x}_u|y_u)$$

- Given the observation \mathbf{X} , the posterior probability is,

$$P(\mathbf{y}|\mathbf{X}) = \frac{P(\mathbf{y})P(\mathbf{X}|\mathbf{y})}{\sum_{\mathbf{y}'} P(\mathbf{y}')P(\mathbf{X}|\mathbf{y}')} \cong \prod_{u \in V} P(\mathbf{x}_u|y_u) \cdot h(y_u) \prod_{(u,v) \in E} H(y_u, y_v)$$

which is just another pairwise MRF.

$\mathbf{h} \in \mathbb{R}^c$, $\mathbf{H} \in \mathbb{R}^{c \times c}$ are model parameters with all positive entries

A STATISTICAL MODEL FOR ATTRIBUTED GRAPHS

- Consider a two-step process for generating node attributes:
 - Sample node labels jointly from a pairwise MRF

$$P(\mathbf{y}) = \frac{\varphi(\mathbf{y})}{\sum_{\mathbf{y}'} \varphi(\mathbf{y}')}, \quad \varphi(\mathbf{y}) = \prod_{u \in V} h(y_u) \prod_{(u,v) \in E} H(y_u, y_v)$$

- Given the node labels, sample the features from $P(\mathbf{X}|\mathbf{y})$

$$P(\mathbf{X}|\mathbf{y}) = \prod_{u \in V} P(\mathbf{x}_u|y_u)$$

- Given the observation \mathbf{X} , the posterior probability is,

$$P(\mathbf{y}|\mathbf{X}) = \frac{P(\mathbf{y})P(\mathbf{X}|\mathbf{y})}{\sum_{\mathbf{y}'} P(\mathbf{y}')P(\mathbf{X}|\mathbf{y}')} \cong \underbrace{\prod_{u \in V} P(\mathbf{x}_u|y_u) \cdot h(y_u)}_{\text{self potential}} \underbrace{\prod_{(u,v) \in E} H(y_u, y_v)}_{\text{pair potential}}$$

which is just another pairwise MRF.

$\mathbf{h} \in \mathbb{R}^c$, $\mathbf{H} \in \mathbb{R}^{c \times c}$ are model parameters with all positive entries

LEARNING ALGO FROM PROBABILISTIC INFERENCE

$$P(\mathbf{y}|\mathbf{X}) \cong \prod_{(u,v) \in E} H(y_u, y_v) \prod_{u \in V} P(\mathbf{x}_u|y_u) \cdot h(y_u)$$

- Probabilistic inference: compute $P(y_u|\mathbf{X})$ with **belief propagation**

LEARNING ALGO FROM PROBABILISTIC INFERENCE

$$P(\mathbf{y}|\mathbf{X}) \cong \prod_{(u,v) \in E} H(y_u, y_v) \prod_{u \in V} P(\mathbf{x}_u|y_u) \cdot h(y_u)$$

- Probabilistic inference: compute $P(y_u|\mathbf{X})$ with **belief propagation**
- Since $P(\mathbf{x}_u|y_u)$, \mathbf{h} , \mathbf{H} are unknown, we need to learn from data

LEARNING ALGO FROM PROBABILISTIC INFERENCE

$$\begin{aligned} P(\mathbf{y}|\mathbf{X}) &\cong \prod_{(u,v) \in E} H(y_u, y_v) \prod_{u \in V} P(\mathbf{x}_u|y_u) \cdot h(y_u) \\ &= \prod_{(u,v) \in E} \textcolor{red}{H}(y_u, y_v) \prod_{u \in V} f_{\theta}(y_u; \mathbf{x}_u) \end{aligned}$$

- Probabilistic inference: compute $P(y_u|\mathbf{X})$ with **belief propagation**
- Since $P(\mathbf{x}_u|y_u)$, \mathbf{h} , \mathbf{H} are unknown, we need to learn from data

LEARNING ALGO FROM PROBABILISTIC INFERENCE

$$\begin{aligned} P(\mathbf{y}|\mathbf{X}) &\cong \prod_{(u,v) \in E} H(y_u, y_v) \prod_{u \in V} P(\mathbf{x}_u|y_u) \cdot h(y_u) \\ &= \prod_{(u,v) \in E} \textcolor{red}{H}(y_u, y_v) \prod_{u \in V} f_{\theta}(y_u; \mathbf{x}_u) \end{aligned}$$

- Probabilistic inference: compute $P(y_u|\mathbf{X})$ with **belief propagation**
- Since $P(\mathbf{x}_u|y_u)$, \mathbf{h} , \mathbf{H} are unknown, we need to learn from data
- This results in a 2-step algorithm,

LEARNING ALGO FROM PROBABILISTIC INFERENCE

$$\begin{aligned} P(\mathbf{y}|\mathbf{X}) &\cong \prod_{(u,v) \in E} H(y_u, y_v) \prod_{u \in V} P(\mathbf{x}_u|y_u) \cdot h(y_u) \\ &= \prod_{(u,v) \in E} \mathbf{H}(y_u, y_v) \prod_{u \in V} f_{\theta}(y_u; \mathbf{x}_u) \end{aligned}$$

- Probabilistic inference: compute $P(y_u|\mathbf{X})$ with **belief propagation**
- Since $P(\mathbf{x}_u|y_u)$, \mathbf{h} , \mathbf{H} are unknown, we need to learn from data
- This results in a 2-step algorithm,
 - compute initial belief on each node u

$$p_u^{(0)} = f_{\theta}(y_u; \mathbf{x}_u) = \text{softmax}(\text{MLP}(\mathbf{x}_u))[y_u]$$

LEARNING ALGO FROM PROBABILISTIC INFERENCE

$$\begin{aligned} P(\mathbf{y}|\mathbf{X}) &\cong \prod_{(u,v) \in E} H(y_u, y_v) \prod_{u \in V} P(\mathbf{x}_u|y_u) \cdot h(y_u) \\ &= \prod_{(u,v) \in E} \mathbf{H}(y_u, y_v) \prod_{u \in V} f_{\theta}(y_u; \mathbf{x}_u) \end{aligned}$$

- Probabilistic inference: compute $P(y_u|\mathbf{X})$ with **belief propagation**
- Since $P(\mathbf{x}_u|y_u)$, \mathbf{h} , \mathbf{H} are unknown, we need to learn from data
- This results in a 2-step algorithm,

- compute initial belief on each node u

$$p_u^{(0)} = f_{\theta}(y_u; \mathbf{x}_u) = \text{softmax}(\text{MLP}(\mathbf{x}_u))[y_u]$$

- run K steps of belief propagation (computed in log-space in practice)

$$p_u^{(k)}(y_u) \cong p_u^{(0)}(y_u) \prod_{v \in N(u)} m_{vu}^{(k)}(y_u)$$

$$m_{vu}^{(k)}(y_u) \cong \sum_{y_v} H_{vu}(y_v, y_u) \cdot p_v^{(k-1)}(y_v) \cdot m_{uv}^{(k-1)}(y_v)^{-1}$$

LEARNING ALGO FROM PROBABILISTIC INFERENCE

$$\begin{aligned} P(\mathbf{y}|\mathbf{X}) &\cong \prod_{(u,v) \in E} H(y_u, y_v) \prod_{u \in V} P(\mathbf{x}_u|y_u) \cdot h(y_u) \\ &= \prod_{(u,v) \in E} \textcolor{red}{H}(y_u, y_v) \prod_{u \in V} f_{\theta}(y_u; \mathbf{x}_u) \end{aligned}$$

- Probabilistic inference: compute $P(y_u|\mathbf{X})$ with **belief propagation**
- Since $P(\mathbf{x}_u|y_u)$, \mathbf{h} , \mathbf{H} are unknown, we need to learn from data
- This results in a 2-step algorithm, which can be trained **end-to-end**

- compute initial belief on each node u

$$p_u^{(0)} = f_{\theta}(y_u; \mathbf{x}_u) = \text{softmax}(\text{MLP}(\mathbf{x}_u))[y_u]$$

- run K steps of belief propagation (computed in log-space in practice)

$$p_u^{(k)}(y_u) \cong p_u^{(0)}(y_u) \prod_{v \in N(u)} m_{vu}^{(k)}(y_u)$$

$$m_{vu}^{(k)}(y_u) \cong \sum_{y_v} H_{vu}(y_v, y_u) \cdot p_v^{(k-1)}(y_v) \cdot m_{uv}^{(k-1)}(y_v)^{-1}$$

LEARNING ALGO FROM PROBABILISTIC INFERENCE

- This results in a 2-step algorithm, which can be trained **end-to-end**

- compute initial belief on each node u

$$p_u^{(0)} = f_{\theta}(y_u; \mathbf{x}_u) = \text{softmax}(\text{MLP}(\mathbf{x}_u))[y_u]$$

- run K steps of belief propagation (computed in log-space in practice)

$$p_u^{(k)}(y_u) \cong p_u^{(0)}(y_u) \prod_{v \in N(u)} m_{vu}^{(k)}(y_u)$$

$$m_{vu}^{(k)}(y_u) \cong \sum_{y_v} H_{vu}(y_v, y_u) \cdot p_v^{(k-1)}(y_v) \cdot m_{uv}^{(k-1)}(y_v)^{-1}$$

- After training, we can infer unknown labels directly $P(y_u | \mathbf{X})$.

LEARNING ALGO FROM PROBABILISTIC INFERENCE

- This results in a 2-step algorithm, which can be trained **end-to-end**

- compute initial belief on each node u

$$p_u^{(0)} = f_{\theta}(y_u; \mathbf{x}_u) = \text{softmax}(\text{MLP}(\mathbf{x}_u))[y_u]$$

- run K steps of belief propagation (computed in log-space in practice)

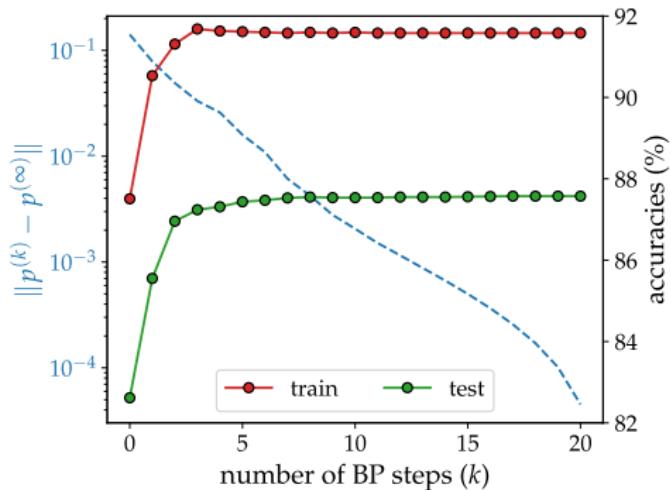
$$p_u^{(k)}(y_u) \cong p_u^{(0)}(y_u) \prod_{v \in N(u)} m_{vu}^{(k)}(y_u)$$

$$m_{vu}^{(k)}(y_u) \cong \sum_{y_v} H_{vu}(y_v, y_u) \cdot p_v^{(k-1)}(y_v) \cdot m_{uv}^{(k-1)}(y_v)^{-1}$$

- After training, we can infer unknown labels directly $P(y_u | \mathbf{X})$.
- Or we can compute $P(y_u | \mathbf{X}, \mathbf{y}_L)$:
further conditioning on \mathbf{y}_L , resulting in a pairwise MRF on $G[U]$

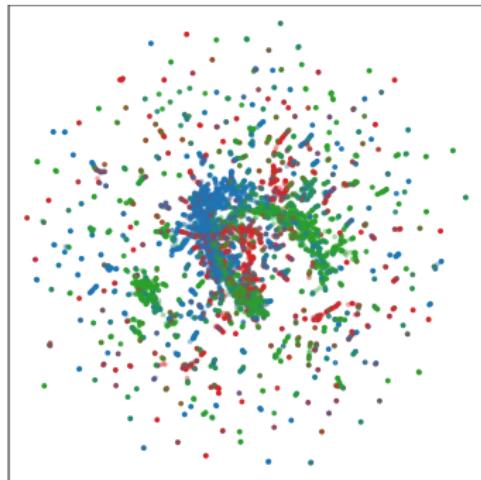
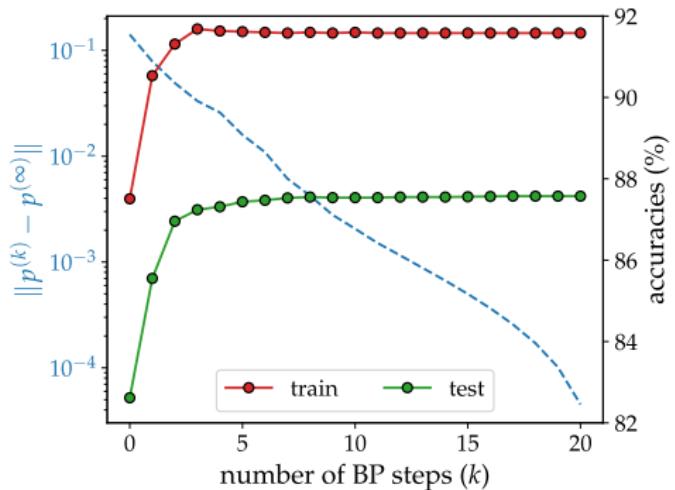
$$P(\mathbf{y}_U | \mathbf{X}, \mathbf{y}_L) \cong \prod_{u \in U} \left(f_{\theta}(y_u; \mathbf{x}_u) \prod_{\substack{u \in U, v \in L, \\ (u, v) \in E}} H_{uv}(y_u, y_v) \right) \prod_{\substack{u, v \in U, \\ (u, v) \in E}} H_{uv}(y_u, y_v)$$

GBPN CASE STUDY ON PUBMED DATASET



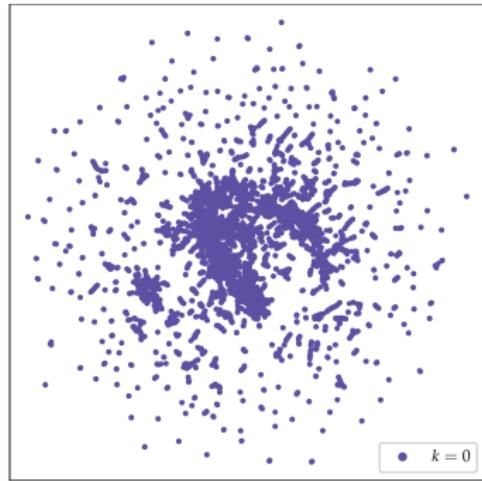
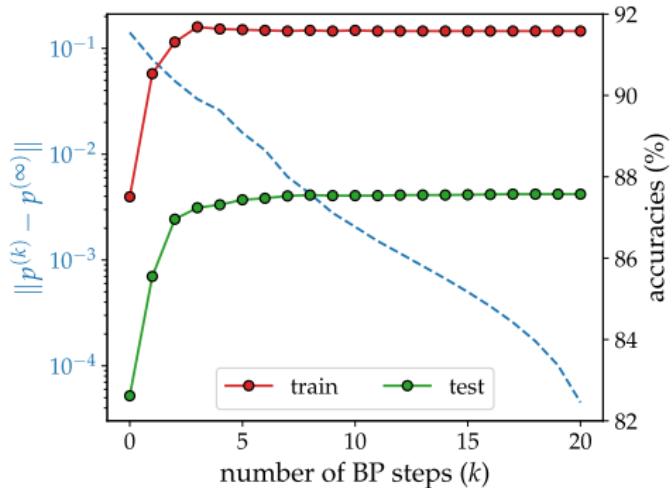
- Belief propagation converge quickly on real-world dataset

GBPN CASE STUDY ON PUBMED DATASET



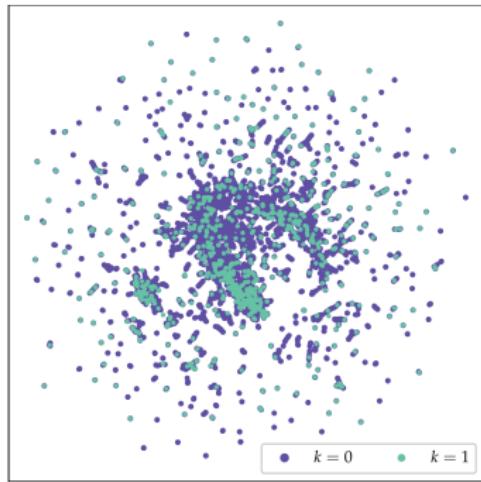
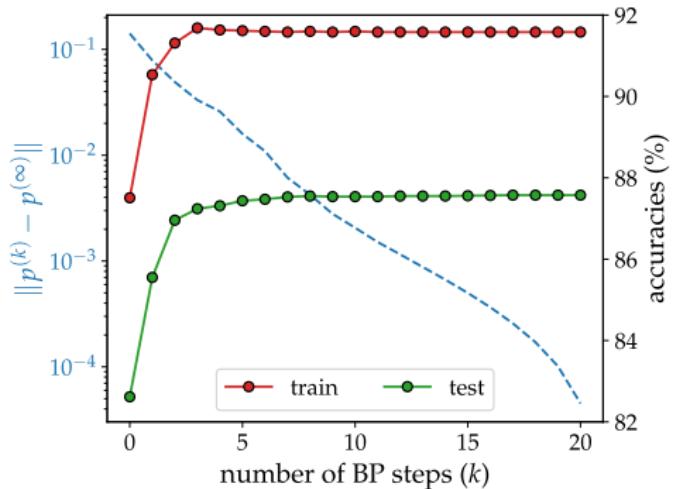
- Belief propagation converge quickly on real-world dataset
- Visualizing ground truth labels on PubMed

GBPN CASE STUDY ON PUBMED DATASET



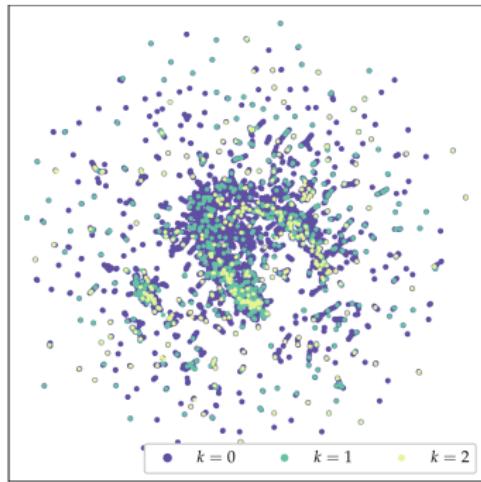
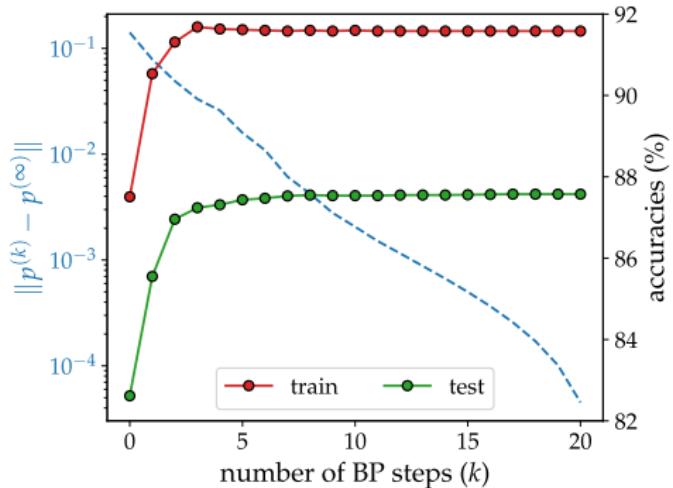
- Belief propagation converge quickly on real-world dataset
- Visualizing ground truth labels on PubMed
- GBPN is interpretable (node correctly predicted by GBPN after k steps)

GBPN CASE STUDY ON PUBMED DATASET



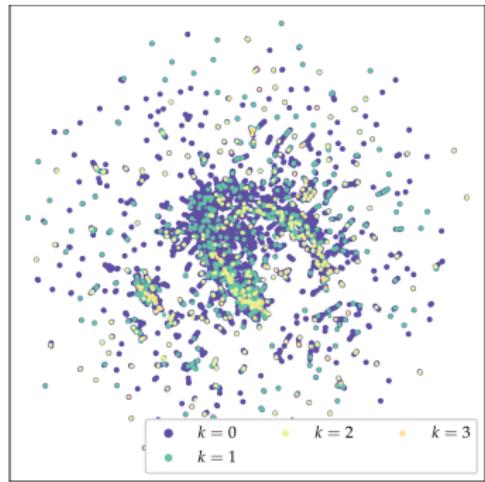
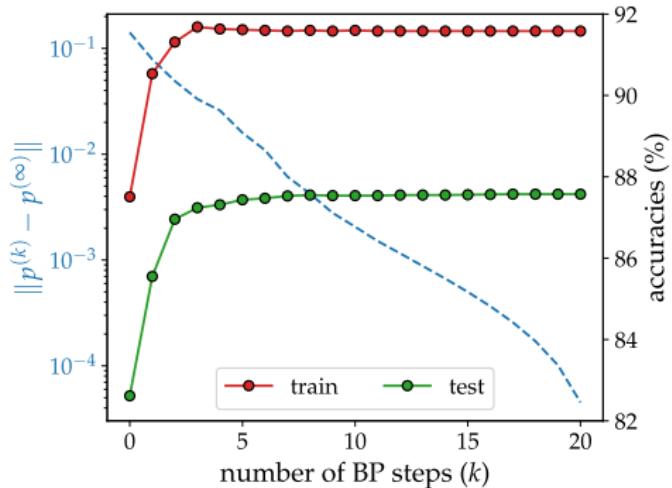
- Belief propagation converge quickly on real-world dataset
- Visualizing ground truth labels on PubMed
- GBPN is interpretable (node correctly predicted by GBPN after k steps)

GBPN CASE STUDY ON PUBMED DATASET



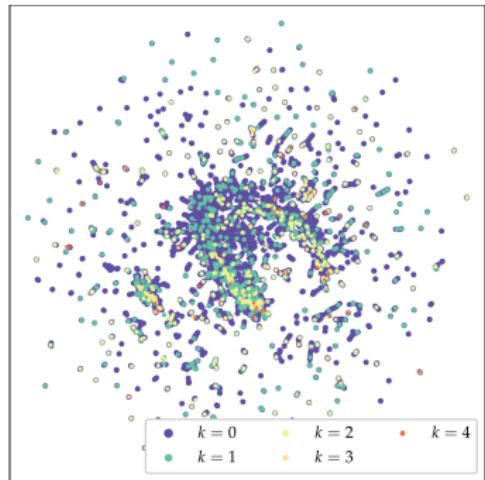
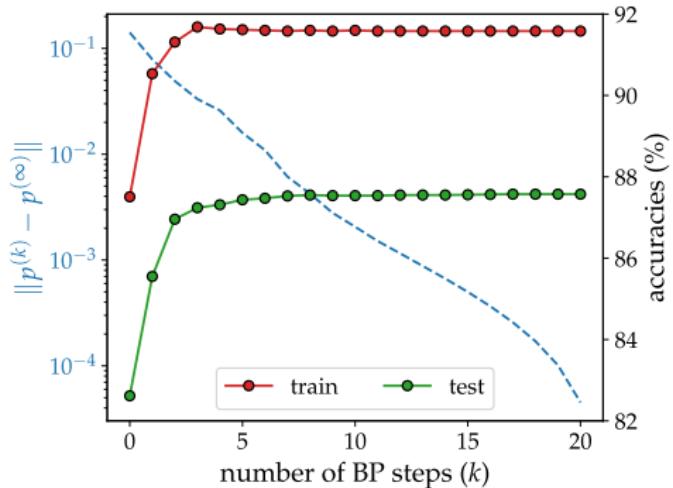
- Belief propagation converge quickly on real-world dataset
- Visualizing ground truth labels on PubMed
- GBPN is interpretable (node correctly predicted by GBPN after k steps)

GBPN CASE STUDY ON PUBMED DATASET



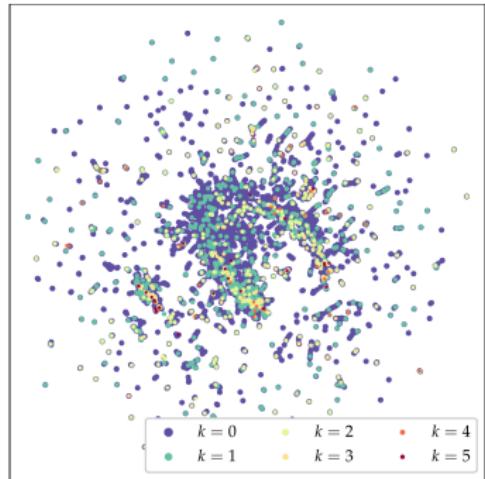
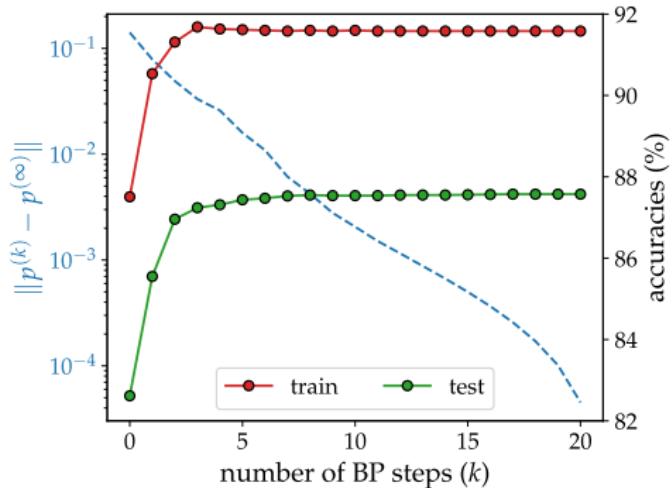
- Belief propagation converge quickly on real-world dataset
- Visualizing ground truth labels on PubMed
- GBPN is interpretable (node correctly predicted by GBPN after k steps)

GBPN CASE STUDY ON PUBMED DATASET



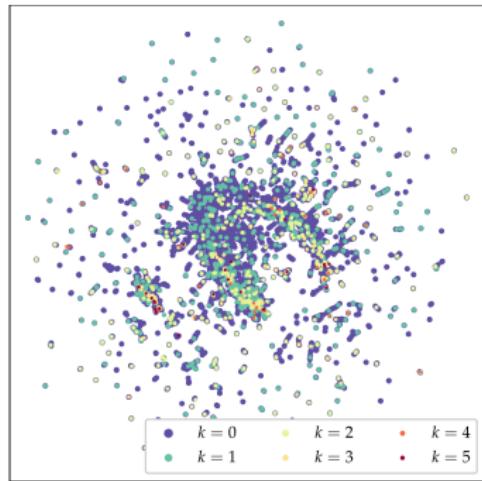
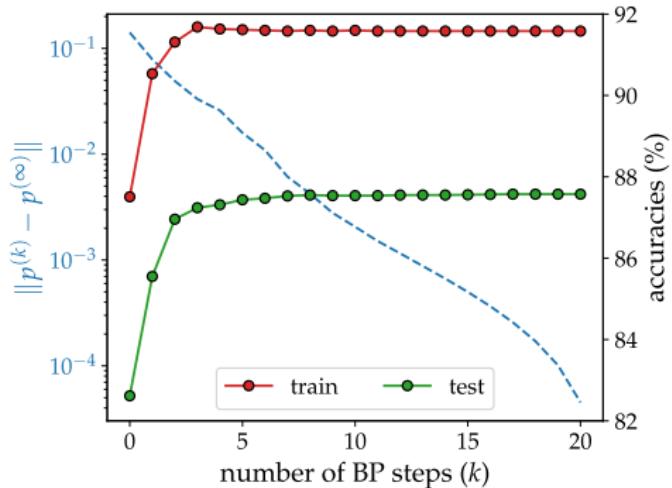
- Belief propagation converge quickly on real-world dataset
- Visualizing ground truth labels on PubMed
- GBPN is interpretable (node correctly predicted by GBPN after k steps)

GBPN CASE STUDY ON PUBMED DATASET



- Belief propagation converge quickly on real-world dataset
- Visualizing ground truth labels on PubMed
- GBPN is interpretable (node correctly predicted by GBPN after k steps)

GBPN CASE STUDY ON PUBMED DATASET



- Belief propagation converge quickly on real-world dataset
- Visualizing ground truth labels on PubMed
- GBPN is interpretable (node correctly predicted by GBPN after k steps)
first get easy examples right, then iteratively correct harder examples with neighbors

PERFORMANCE ON BENCHMARK DATASETS

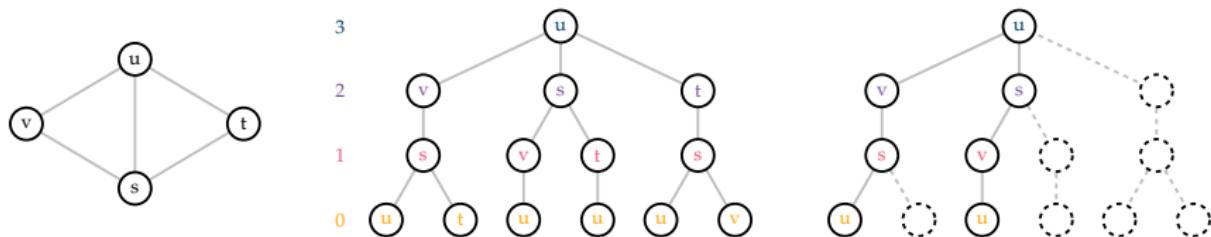
dataset	MLP	SGC	GCN	GraphSAGE	GAT	GBPN(I)	GBPN(T)
County	89.8 ± 0.6	88.2 ± 0.7	87.9 ± 0.7	90.9 ± 0.5	90.5 ± 0.4	90.1 ± 0.7	90.4 ± 0.8
Sexual	73.9 ± 1.4	76.1 ± 1.4	83.7 ± 1.1	93.0 ± 0.9	93.7 ± 0.9	97.0 ± 0.5	97.1 ± 0.5
Cora	72.4 ± 1.1	87.1 ± 0.7	87.1 ± 0.7	87.1 ± 0.8	86.7 ± 0.7	84.8 ± 0.8	84.8 ± 0.8
CiteSeer	70.5 ± 0.9	72.9 ± 1.1	73.4 ± 0.9	73.1 ± 0.8	72.5 ± 0.8	73.9 ± 0.7	73.7 ± 0.7
PubMed	86.6 ± 0.4	86.9 ± 0.3	87.0 ± 0.3	87.8 ± 0.3	88.0 ± 0.2	88.2 ± 0.3	88.2 ± 0.3
CS	94.1 ± 0.2	93.1 ± 0.2	93.1 ± 0.3	93.6 ± 0.3	94.3 ± 0.3	95.4 ± 0.1	95.4 ± 0.2
Physics	95.8 ± 0.2	96.1 ± 0.1	96.1 ± 0.1	96.2 ± 0.2	96.4 ± 0.1	96.8 ± 0.1	96.8 ± 0.1

PERFORMANCE ON BENCHMARK DATASETS

dataset	MLP	SGC	GCN	GraphSAGE	GAT	GBPN(I)	GBPN(T)
County	89.8 ± 0.6	88.2 ± 0.7	87.9 ± 0.7	90.9 ± 0.5	90.5 ± 0.4	90.1 ± 0.7	90.4 ± 0.8
Sexual	73.9 ± 1.4	76.1 ± 1.4	83.7 ± 1.1	93.0 ± 0.9	93.7 ± 0.9	97.0 ± 0.5	97.1 ± 0.5
Cora	72.4 ± 1.1	87.1 ± 0.7	87.1 ± 0.7	87.1 ± 0.8	86.7 ± 0.7	84.8 ± 0.8	84.8 ± 0.8
CiteSeer	70.5 ± 0.9	72.9 ± 1.1	73.4 ± 0.9	73.1 ± 0.8	72.5 ± 0.8	73.9 ± 0.7	73.7 ± 0.7
PubMed	86.6 ± 0.4	86.9 ± 0.3	87.0 ± 0.3	87.8 ± 0.3	88.0 ± 0.2	88.2 ± 0.3	88.2 ± 0.3
CS	94.1 ± 0.2	93.1 ± 0.2	93.1 ± 0.3	93.6 ± 0.3	94.3 ± 0.3	95.4 ± 0.1	95.4 ± 0.2
Physics	95.8 ± 0.2	96.1 ± 0.1	96.1 ± 0.1	96.2 ± 0.2	96.4 ± 0.1	96.8 ± 0.1	96.8 ± 0.1

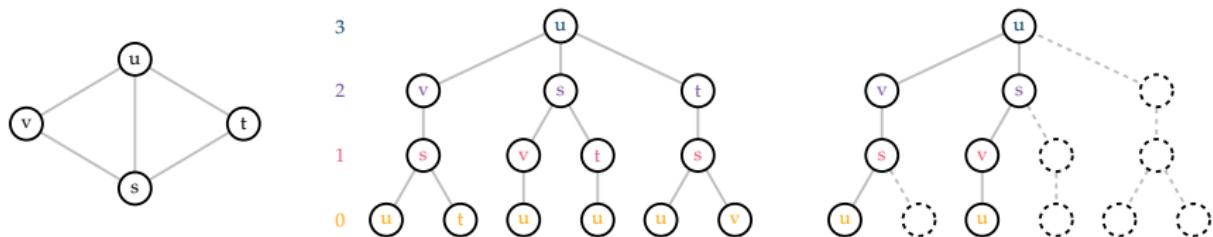
- GBPN is not only more interpretable, but also more accurate

PERFORMANCE ON LARGE-SCALE DATASETS



- We design a subsampling algorithm for mini-batch training

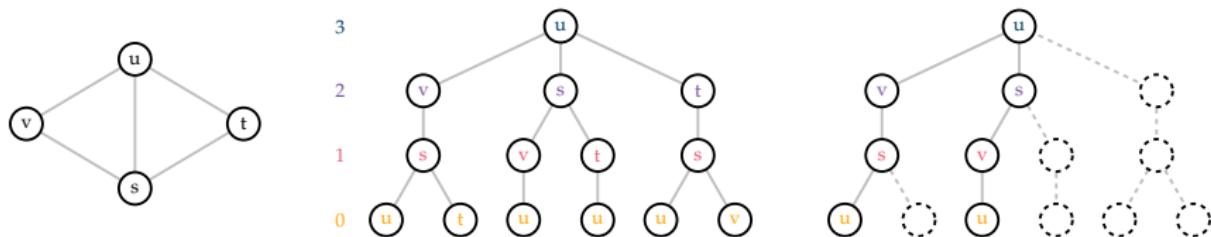
PERFORMANCE ON LARGE-SCALE DATASETS



dataset	MLP	GraphSAGE	GAT	GBPN(I)	GBPN(T)
Elliptic	98.2 ± 0.2	98.4 ± 0.2	98.2 ± 0.3	98.4 ± 0.1	98.4 ± 0.2
Chase	61.1 ± 0.3	77.5 ± 0.5	74.6 ± 0.8	83.0 ± 0.5	86.2 ± 0.5
arXiv	54.6 ± 0.3	70.4 ± 0.3	70.4 ± 0.2	69.4 ± 1.1	70.1 ± 0.1
Products	61.5 ± 0.2	78.1 ± 0.2	76.8 ± 0.0	81.4 ± 0.4	81.2 ± 0.1

- We design a subsampling algorithm for mini-batch training

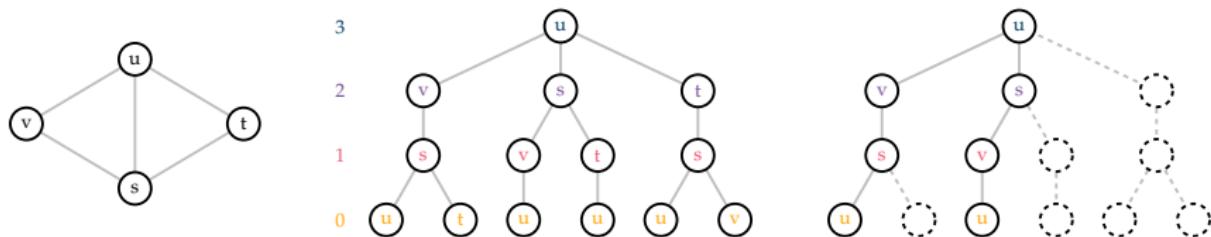
PERFORMANCE ON LARGE-SCALE DATASETS



dataset	MLP	GraphSAGE	GAT	GBPN(I)	GBPN(T)
Elliptic	98.2 ± 0.2	98.4 ± 0.2	98.2 ± 0.3	98.4 ± 0.1	98.4 ± 0.2
Chase	61.1 ± 0.3	77.5 ± 0.5	74.6 ± 0.8	83.0 ± 0.5	86.2 ± 0.5
arXiv	54.6 ± 0.3	70.4 ± 0.3	70.4 ± 0.2	69.4 ± 1.1	70.1 ± 0.1
Products	61.5 ± 0.2	78.1 ± 0.2	76.8 ± 0.0	81.4 ± 0.4	81.2 ± 0.1

- We design a subsampling algorithm for mini-batch training
- GBPN outperforms baselines overall

PERFORMANCE ON LARGE-SCALE DATASETS



dataset	MLP	GraphSAGE	GAT	GBPN(I)	GBPN(T)
Elliptic	98.2 ± 0.2	98.4 ± 0.2	98.2 ± 0.3	98.4 ± 0.1	98.4 ± 0.2
Chase	61.1 ± 0.3	77.5 ± 0.5	74.6 ± 0.8	83.0 ± 0.5	86.2 ± 0.5
arXiv	54.6 ± 0.3	70.4 ± 0.3	70.4 ± 0.2	69.4 ± 1.1	70.1 ± 0.1
Products	61.5 ± 0.2	78.1 ± 0.2	76.8 ± 0.0	81.4 ± 0.4	81.2 ± 0.1

- We design a subsampling algorithm for mini-batch training
- GBPN outperforms baselines overall
- GBPN(T) outperforms GBPN(I) by a noticeable margin on some dataset

CONCLUSIONS

- One key aspect of semi-supervised node regression is to leverage feature correlation and attribute homophily.

CONCLUSIONS

- One key aspect of semi-supervised node regression is to leverage feature correlation and attribute homophily.
- My thesis proposed a generative model that accounts for three different types of correlations in attributed graphs. It unifies existing algorithms and motivates new ones with the state-of-the-art performance.

CONCLUSIONS

- One key aspect of semi-supervised node regression is to leverage feature correlation and attribute homophily.
- My thesis proposed a generative model that accounts for three different types of correlations in attributed graphs. It unifies existing algorithms and motivates new ones with the state-of-the-art performance.
- Our model successfully generalization to non-homophily graphs and node classification setting.

ACKNOWLEDGMENTS

- **Committee Members**

- Prof. Austin Benson
- Prof. David Bindel
- Prof. Jon Kleinberg



- **Collaborators**

- Prof. Michael Schaub
- Prof. Santiago Segarra



- **Supportors**

- Parents & Family
- Friends & Stacey



MY RESEARCH

- **Modeling and Inferring Attributed Graphs**

- Residual Correlation in Graph Neural Network Regression,
Junteng Jia, Austin R. Benson, KDD2020,
<https://github.com/000Justin000/gnn-residual-correlation>
- A Unifying Generative Model for Graph Learning Algorithms,
Junteng Jia, Austin R. Benson, arXiv2021,
<https://github.com/000Justin000/GaussianMRF>
- Graph Belief Propagation Networks,
Junteng Jia, Austin R. Benson, (in preparation)

MY RESEARCH

- **Modeling and Inferring Attributed Graphs**

- Residual Correlation in Graph Neural Network Regression,
Junteng Jia, Austin R. Benson, KDD2020,
<https://github.com/000Justin000/gnn-residual-correlation>
- A Unifying Generative Model for Graph Learning Algorithms,
Junteng Jia, Austin R. Benson, arXiv2021,
<https://github.com/000Justin000/GaussianMRF>
- Graph Belief Propagation Networks,
Junteng Jia, Austin R. Benson, (in preparation)

- **Find me at ...**

- Full list of publications: [Google Scholar](#)
- Personal website: <https://000justin000.github.io>

MY RESEARCH

- **Modeling and Inferring Attributed Graphs**

- Residual Correlation in Graph Neural Network Regression,
Junteng Jia, Austin R. Benson, KDD2020,
<https://github.com/000Justin000/gnn-residual-correlation>
- A Unifying Generative Model for Graph Learning Algorithms,
Junteng Jia, Austin R. Benson, arXiv2021,
<https://github.com/000Justin000/GaussianMRF>
- Graph Belief Propagation Networks,
Junteng Jia, Austin R. Benson, (in preparation)

- **Find me at ...**

- Full list of publications: [Google Scholar](#)
- Personal website: <https://000justin000.github.io>

- **My researches are supported by ...**

