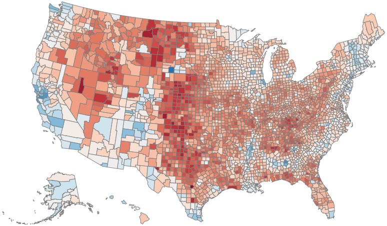
Modeling and Inferring Attributed Graphs



Junteng Jia

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

Ph.D. Thesis Defense – April 12, 2021

Research Overview

Physics/Dynamics

- Quantum ESPRESSO, JPCM
- Polymer flexibility, Nat. Comm.
- Hybrid DFT, JCTC

- Neural jump SDE, Neurips19

Sci. comp. | ML

- Core-periphery, WSDM19
- Edge-flow learning, KDD19

- GNN residual correlation, KDD20
- Unifying generative model, arXiv21
- Graph belief propagation network

Graphs

Research Overview

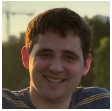
Physics/Dynamics

- Quantum ESPRESSO, JPCM
- Polymer flexibility, Nat. Comm.
- Hybrid DFT, JCTC
- Neural jump SDE, Neurips19

Sci. comp. ML

- Core-periphery, WSDM19
- Edge-flow learning, KDD19
- GNN residual correlation, KDD20
- Unifying generative model, arXiv21
- Graph belief propagation network

Graphs



Research Overview

Physics/Dynamics

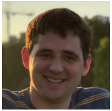
- Quantum ESPRESSO, JPCM
- Polymer flexibility, Nat. Comm.
- Hybrid DFT, JCTC

- Neural jump SDE, Neurips19

Sci. comp. ML

- Core-periphery, WSDM19
- Edge-flow learning, KDD19

- GNN residual correlation, KDD20
- Unifying generative model, arXiv21
- Graph belief propagation network



Graphs

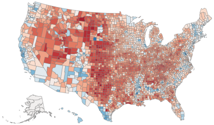
Graphs Data are Increasingly More Complex

Social Networks

nodes: people
edges: friendships

Finance

nodes: accounts
edges: transations



Politics

nodes: regions
edges: adjacency

Traffic

nodes: stations
edges: railways

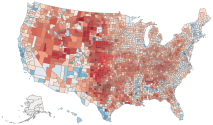
Graphs Data are Increasingly More Complex

Social Networks

nodes: people
edges: friendships

Finance

nodes: accounts
edges: transations



Politics

nodes: regions
edges: adjacency

Traffic

nodes: stations
edges: railways

- node attributes contains rich information,

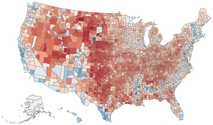
Graphs Data are Increasingly More Complex

Social Networks

nodes: people
edges: friendships

Finance

nodes: accounts
edges: transations



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, . . .

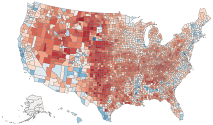
Graphs Data are Increasingly More Complex

Social Networks

nodes: people
edges: friendships

Finance

nodes: accounts
edges: transations



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, . . .

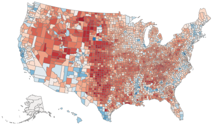
Graphs Data are Increasingly More Complex

Social Networks

nodes: people
edges: friendships

Finance

nodes: accounts
edges: transations



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

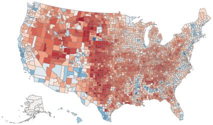
We Often Want to Predict Node Labels

Social Networks

nodes: people
edges: friendships

Finance

nodes: accounts
edges: transations



Politics

nodes: regions
edges: adjacency

Traffic

nodes: stations
edges: railways

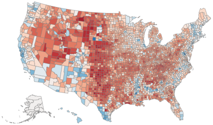
We Often Want to Predict Node Labels

Social Networks

nodes: people
edges: friendships

Finance

nodes: accounts
edges: transations



Politics

nodes: regions
edges: adjacency

Traffic

nodes: stations
edges: railways

- predicting missing node attributes has many real-world applications

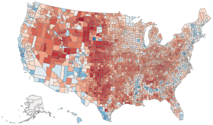
We Often Want to Predict Node Labels

Social Networks

nodes: people
edges: friendships

Finance

nodes: accounts
edges: transations



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]

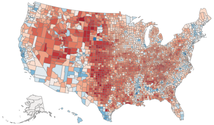
We Often Want to Predict Node Labels

Social Networks

nodes: people
edges: friendships

Finance

nodes: accounts
edges: transations



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

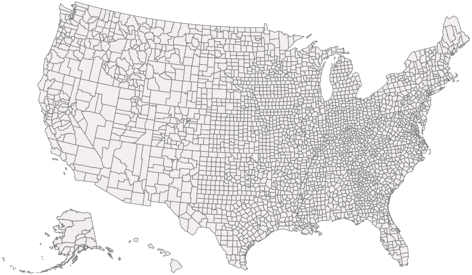
- First, we develop theory and algorithms on regression problems
  Outcomes are real-valued like age, income, temperature ...

Regression vs Classification

- First, we develop theory and algorithms 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

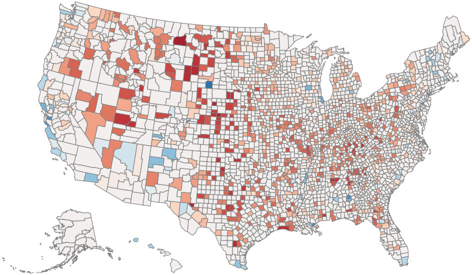
Motivating example — Predicting election from pollsters



- Each county as a node; bordering counties are connected

Node Label Prediction in Attributed Graphs

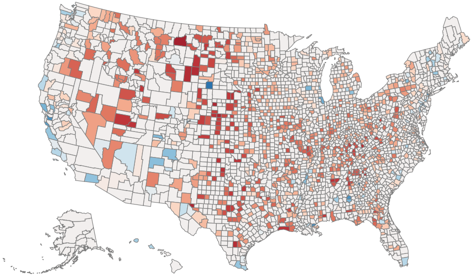
Motivating example — Predicting election from pollsters



- 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

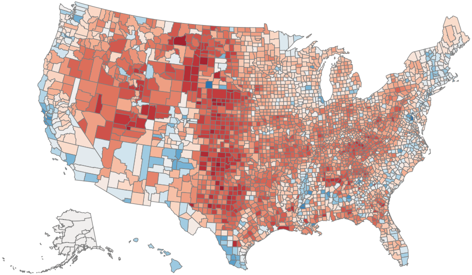
Motivating example — Predicting election from pollsters



- 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

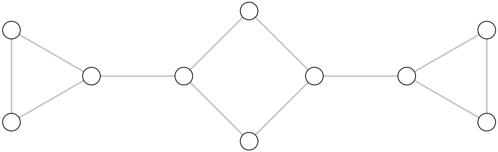
Motivating example — Predicting election from pollsters



- 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

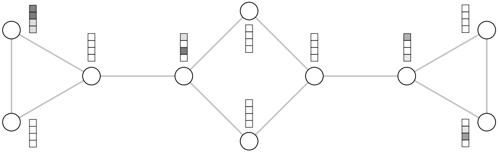
Problem statement



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

Node Label Prediction in Attributed Graphs

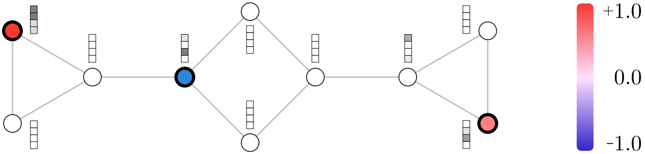
Problem statement



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

Node Label Prediction in Attributed Graphs

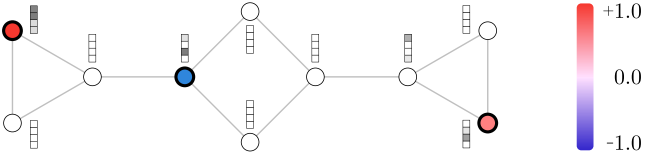
Problem statement



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

Node Label Prediction in Attributed Graphs

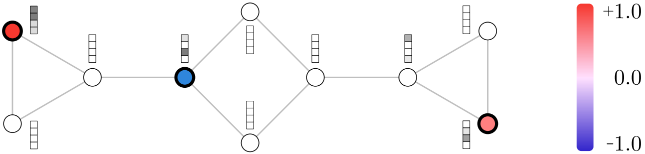
Problem statement



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

There are two Broad Classes of Methods

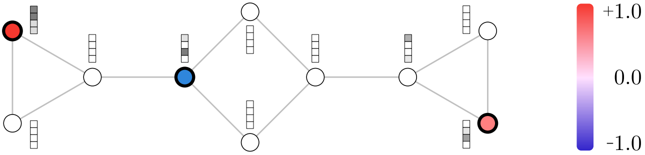
Label propagation is a transductive method [early 2000s]



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

There are two Broad Classes of Methods

Label propagation is a transductive method [early 2000s]

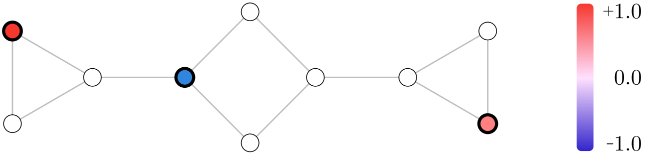


- assumption: connected nodes have similar labels

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

There are two Broad Classes of Methods

Label propagation is a transductive method [early 2000s]

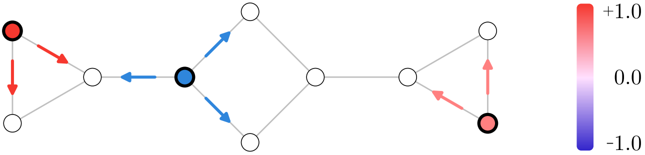


- assumption: connected nodes have similar labels
  - ignore vertex features,
  - set initial prediction $\hat{\boldsymbol{y}}$: $\hat{y}_u = y_u$ if observed, otherwise $\hat{y}_u = 0$

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

There are two Broad Classes of Methods

Label propagation is a transductive method [early 2000s]

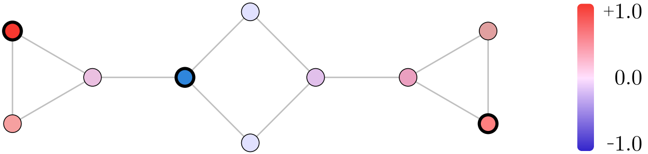


- assumption: connected nodes have similar labels
  - ignore vertex features, diffuse labels along the edges
  - set initial prediction $\hat{\boldsymbol{y}}$: $\hat{y}_u = y_u$ if observed, otherwise $\hat{y}_u = 0$
  - update: $\boldsymbol{y}^{(k)} = \alpha \cdot \boldsymbol{S}\boldsymbol{y}^{(k-1)}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

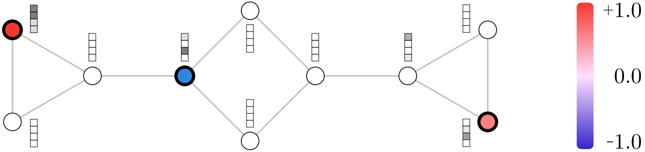


- assumption: connected nodes have similar labels
  - ignore vertex features, diffuse labels along the edges
  - set initial prediction $\hat{\boldsymbol{y}}$: $\hat{y}_u = y_u$ if observed, otherwise $\hat{y}_u = 0$
  - update: $\boldsymbol{y}^{(k)} = \alpha \cdot \boldsymbol{S}\boldsymbol{y}^{(k-1)}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

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

There are two Broad Classes of Methods

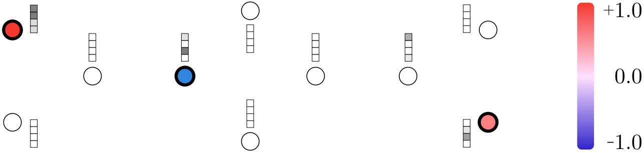
Inductive methods start with naive i.i.d. predictors



- assumption: a label only depend on local features

There are two Broad Classes of Methods

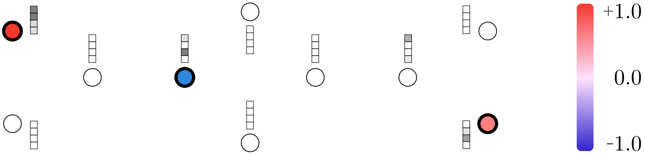
Inductive methods start with naive i.i.d. predictors



- 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

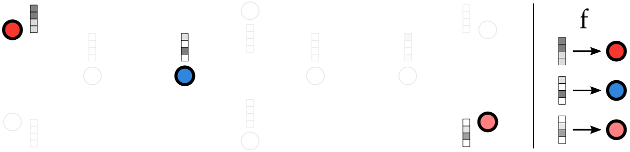
Inductive methods start with naive i.i.d. predictors



- 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

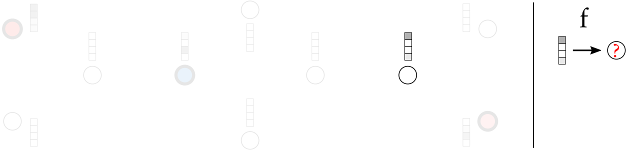
Inductive methods start with naive i.i.d. predictors



- 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

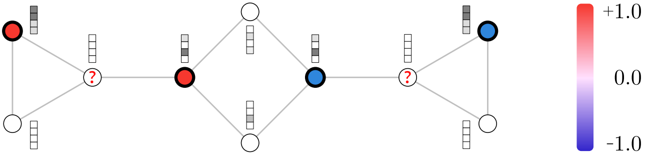
Inductive methods start with naive i.i.d. predictors



- 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 $f : \mathcal{X} \to \mathcal{Y}$; inference: $\hat{y}_u = f(\boldsymbol{x}_u)$

There are two Broad Classes of Methods

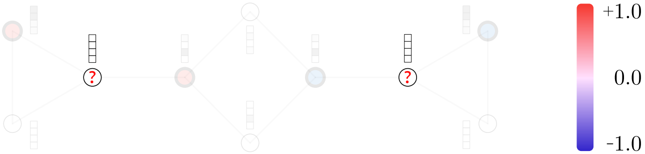
Inductive methods start with naive i.i.d. predictors



- 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 $f : \mathcal{X} \to \mathcal{Y}$; inference: $\hat{y}_u = f(\boldsymbol{x}_u)$
- problem: correlations between neighbors NOT considered

There are two Broad Classes of Methods

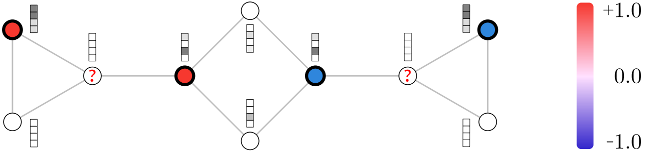
Inductive methods start with naive i.i.d. predictors



- 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 $f : \mathcal{X} \to \mathcal{Y}$; inference: $\hat{y}_u = f(\boldsymbol{x}_u)$
- problem: correlations between neighbors NOT considered
  - vertices with the same features are indistinguishable

There are two Broad Classes of Methods

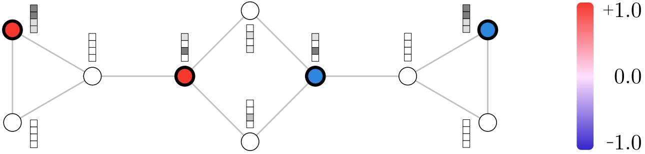
Inductive methods start with naive i.i.d. predictors



- 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 $f: \mathcal{X} \to \mathcal{Y}$; inference: $\hat{y}_u = f(\boldsymbol{x}_u)$
- 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]

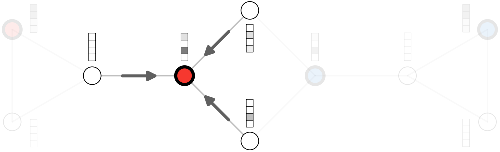


- assumption: a label only depend on features in the neighborhood

[Kipf-Welling 16; Hamilton+ 17; Zhou+ 18; ∼10,000 papers in 5 years]

There are two Broad Classes of Methods

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



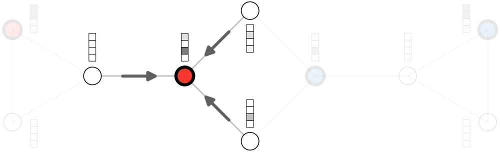
- assumption: a label only depend on features in the neighborhood

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

[Kipf-Welling 16; Hamilton+ 17; Zhou+ 18; ∼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

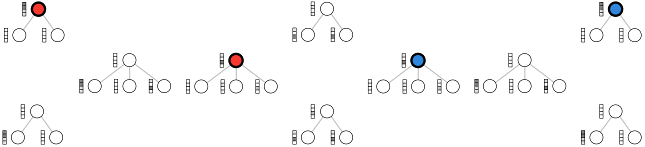
$$\boldsymbol{h}_u^{(0)} = \boldsymbol{x}_u; \;\; \boldsymbol{h}_u^{(k)} = q\left(\boldsymbol{h}_u^{(k-1)}, \{\boldsymbol{h}_v^{(k-1)}\}_{v \in N(u)}\right); \;\; \hat{y}_u = g(\boldsymbol{h}_u^{(K)})$$

  - prediction decided by subtree rooted at u

[Kipf-Welling 16; Hamilton+ 17; Zhou+ 18; ∼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

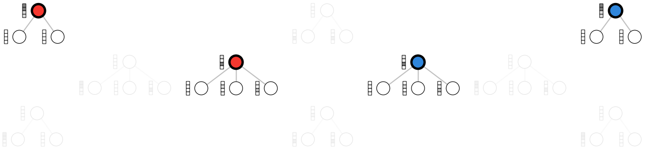
$$\boldsymbol{h}_u^{(0)} = \boldsymbol{x}_u; \;\; \boldsymbol{h}_u^{(k)} = q\left(\boldsymbol{h}_u^{(k-1)}, \{\boldsymbol{h}_v^{(k-1)}\}_{v \in N(u)}\right); \;\; \hat{y}_u = g(\boldsymbol{h}_u^{(K)})$$

  - prediction decided by subtree rooted at u

[Kipf-Welling 16; Hamilton+ 17; Zhou+ 18; ∼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

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

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

[Kipf-Welling 16; Hamilton+ 17; Zhou+ 18; ∼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

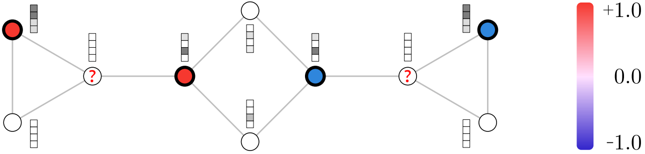
$$\boldsymbol{h}_u^{(0)} = \boldsymbol{x}_u; \;\; \boldsymbol{h}_u^{(k)} = q\left(\boldsymbol{h}_u^{(k-1)}, \{\boldsymbol{h}_v^{(k-1)}\}_{v \in N(u)}\right); \;\; \hat{y}_u = g(\boldsymbol{h}_u^{(K)})$$

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

[Kipf-Welling 16; Hamilton+ 17; Zhou+ 18; ∼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

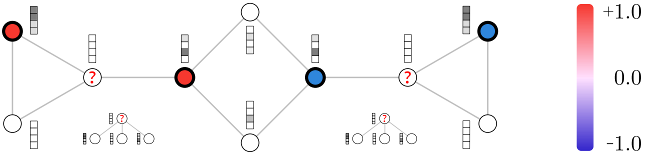
$$\boldsymbol{h}_u^{(0)} = \boldsymbol{x}_u; \;\; \boldsymbol{h}_u^{(k)} = q\left(\boldsymbol{h}_u^{(k-1)}, \{\boldsymbol{h}_v^{(k-1)}\}_{v \in N(u)}\right); \;\; \hat{y}_u = g(\boldsymbol{h}_u^{(K)})$$

  - prediction decided by subtree rooted at u
  - learn a function $\hat{y}_u = f_\theta\left(\{\boldsymbol{x}_v\}_{v \in N_K(u)}\right)$; inductive inference
- problem: correlations between neighboring labels NOT considered

[Kipf-Welling 16; Hamilton+ 17; Zhou+ 18; ∼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

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

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

[Kipf-Welling 16; Hamilton+ 17; Zhou+ 18; ∼10,000 papers in 5 years]

Combing GNN with Label Propagation

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

Combing 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

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

Can we combine an inductive GNN with Label Propagation?

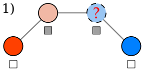
- decompose outcomes as learnable from features and correlated residuals

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

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

Combing GNN with Label Propagation

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



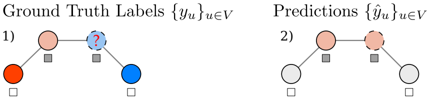
- decompose outcomes as learnable from features and correlated residuals

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

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

- sequentially deploy GNN and label propagation

Combing GNN with Label Propagation



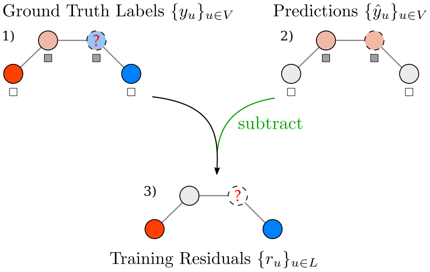
- decompose outcomes as learnable from features and correlated residuals

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

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

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

Combing GNN with Label Propagation



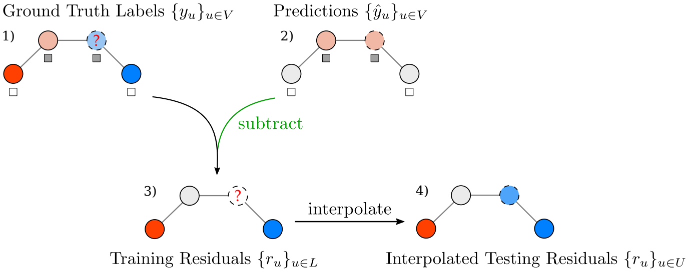
- decompose outcomes as learnable from features and correlated residuals

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

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

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

Combing GNN with Label Propagation



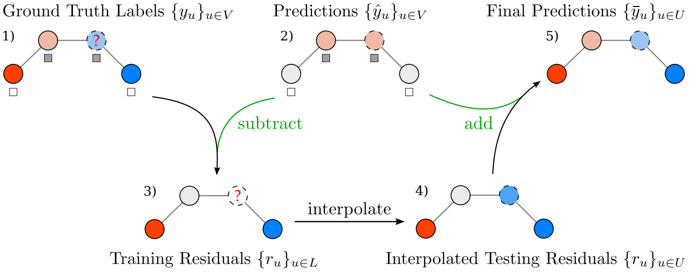
- decompose outcomes as learnable from features and correlated residuals

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

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

Combing GNN with Label Propagation



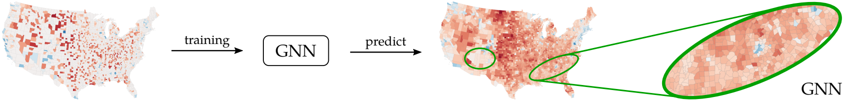
- decompose outcomes as learnable from features and correlated residuals

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

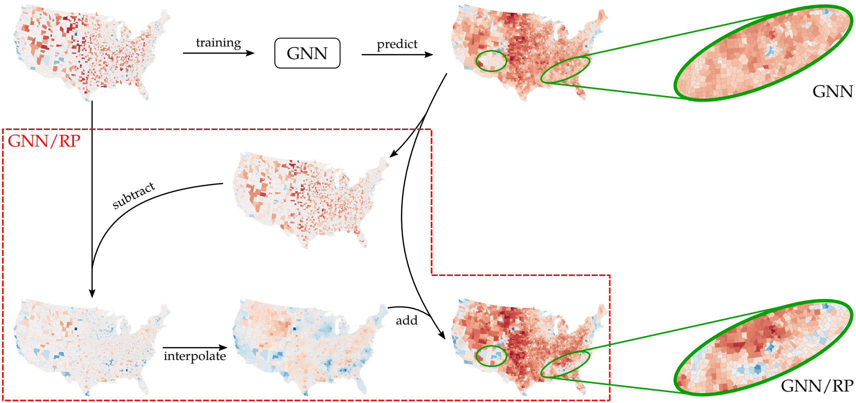
conceptually similar to boosting; the two "weak predictors" uses different inputs

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

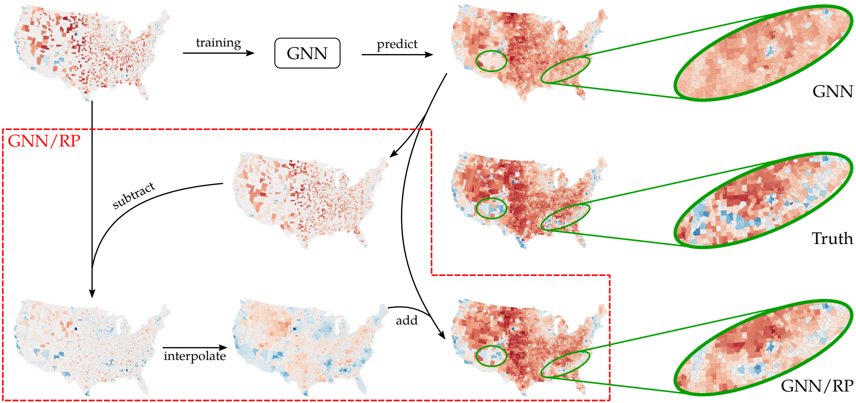
GNN/RP Works Super Well in Practice



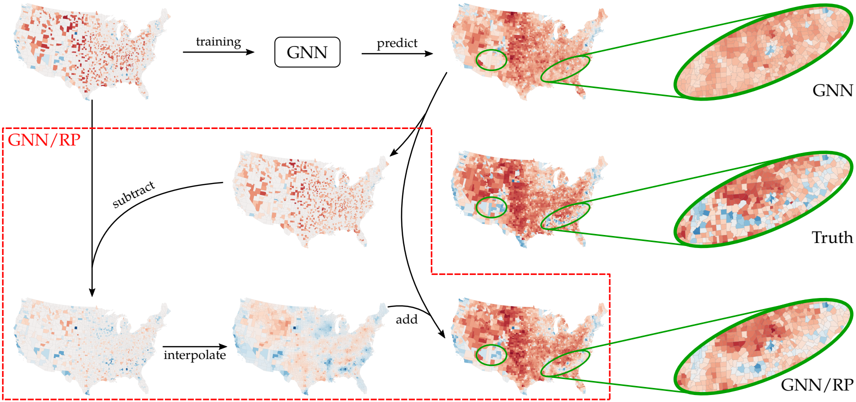
GNN/RP Works Super Well in Practice



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

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

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

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

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

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

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

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

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

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

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

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

- residual propagation assumes correlated errors $\boldsymbol{r} \sim \mathcal{N}\left(\boldsymbol{0}, (\boldsymbol{I} - \alpha \boldsymbol{S})^{-1}\right)$
  - $\boldsymbol{S} = \boldsymbol{D}^{-1/2} \boldsymbol{W} \boldsymbol{D}^{-1/2}$ 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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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, $\boldsymbol{X} = [\boldsymbol{x}_u]_{u \in V}$
  - labels on a subset of vertices $L \subset V$, training labels $\boldsymbol{y}_L = [y_u]_{u \in L}$
- Problem output:
  - labels on the rest of vertices $U \equiv V \backslash L$, testing labels $\boldsymbol{y}_U = [y_u]_{u \in U}$

Statistical Model for Node Attributes

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

Statistical Model for Node Attributes

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

Statistical Model for Node Attributes

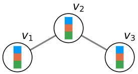
- Problem input:
  - graph topology $G(V, E)$
  - features on all vertices, $\boldsymbol{X} = [\boldsymbol{x}_u]_{u \in V}$
  - labels on a subset of vertices $L \subset V$, training labels $\boldsymbol{y}_L = [y_u]_{u \in L}$
- Problem output:
  - labels on the rest of vertices $U \equiv V \backslash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

Statistical Model for Node Attributes

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

Statistical Model for Node Attributes

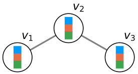
Data Type | Corresponding Gaussian MRF | Learning Algorithm



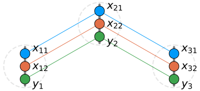
graph data ($\mathbf{X}$, $\mathbf{y}$, G)

Statistical Model for Node Attributes

Data Type	Corresponding Gaussian MRF	Learning Algorithm



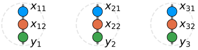
graph data ($\mathbf{X}$, $\mathbf{y}$, G)

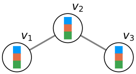


Statistical Model for Node Attributes

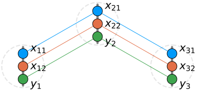
Data Type | Corresponding Gaussian MRF | Learning Algorithm

i.i.d. data (**X, y**)





graph data (**X**, **y**, G)



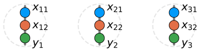
Statistical Model for Node Attributes

Data Type	Corresponding Gaussian MRF		Learning Algorithm

Data Type

i.i.d. data (**X**, **y**)

Corresponding Gaussian MRF

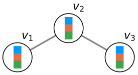


$\xrightarrow[\text{e.g. } L=\{1,3\},\ U=\{2\}]{\text{condition on } \mathbf{X}}$

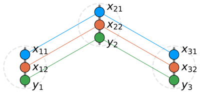
Learning Algorithm

linear regression

$E[\mathbf{y}_U|\mathbf{X}] = \mathbf{X}_U\beta \qquad \beta = (\mathbf{X}_L^\intercal \mathbf{X}_L)^{-1}\mathbf{X}_L^\intercal \mathbf{y}_L$



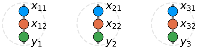
graph data (**X**, **y**, G)



Statistical Model for Node Attributes

Data Type | Corresponding Gaussian MRF | Learning Algorithm

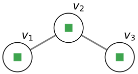
i.i.d. data ($\mathbf{X}$, $\mathbf{y}$)



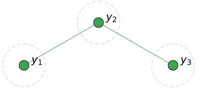
condition on $\mathbf{X}$ → *e.g. L={1,3}, U={2}*

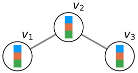
linear regression

$E[\mathbf{y}_U|\mathbf{X}] = \mathbf{X}_U\beta \qquad \beta = (\mathbf{X}_L^\top\mathbf{X}_L)^{-1}\mathbf{X}_L^\top\mathbf{y}_L$

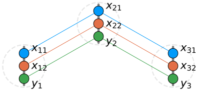


graph data ($\mathbf{y}$, G)



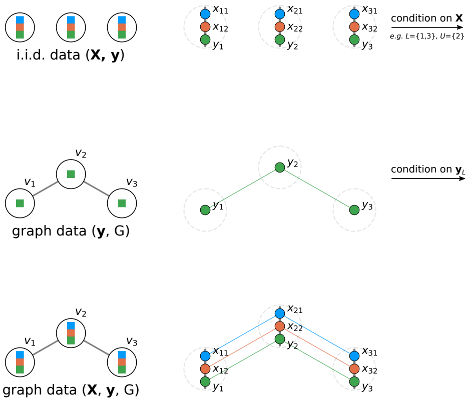


graph data ($\mathbf{X}$, $\mathbf{y}$, G)



Statistical Model for Node Attributes

Data Type | Corresponding Gaussian MRF | Learning Algorithm



linear regression

$E[\mathbf{y}_U|\mathbf{X}] = \mathbf{X}_U\beta \qquad \beta = (\mathbf{X}_L^\top\mathbf{X}_L)^{-1}\mathbf{X}_L^\top\mathbf{y}_L$

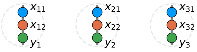
label propagation

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

Statistical Model for Node Attributes

Data Type	Corresponding Gaussian MRF		Learning Algorithm

i.i.d. data ($\mathbf{X}$, $\mathbf{y}$)

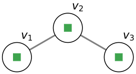


condition on $\mathbf{X}$

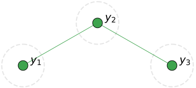
e.g. $L=\{1,3\}$, $U=\{2\}$

linear regression

$E[\mathbf{y}_U|\mathbf{X}] = \mathbf{X}_U\beta \qquad \beta = (\mathbf{X}_L^\top\mathbf{X}_L)^{-1}\mathbf{X}_L^\top\mathbf{y}_L$



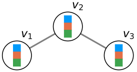
graph data ($\mathbf{y}$, G)



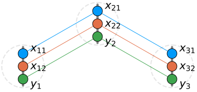
condition on $\mathbf{y}_L$

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)



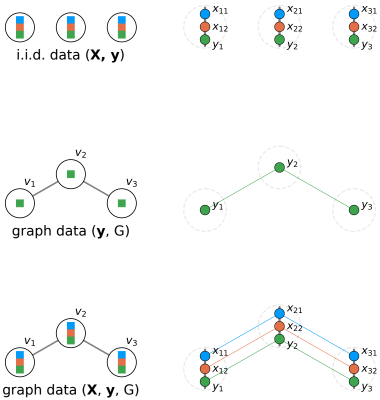
condition on $\mathbf{X}$

linear GCN

$E[\mathbf{y}_U|\mathbf{X}] = [(\mathbf{I}+\omega\mathbf{N})^{-1}\mathbf{X}\beta]_U$

Statistical Model for Node Attributes

Data Type — Corresponding Gaussian MRF — Learning Algorithm



$\xrightarrow[\text{e.g. } L=\{1,3\},\ U=\{2\}]{\text{condition on } \mathbf{X}}$ linear regression

$E[\mathbf{y}_U|\mathbf{X}] = \mathbf{X}_U\beta \qquad \beta = (\mathbf{X}_L^\top\mathbf{X}_L)^{-1}\mathbf{X}_L^\top\mathbf{y}_L$

$\xrightarrow{\text{condition on } \mathbf{y}_L}$ label propagation

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

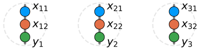
$\xrightarrow{\text{condition on } \mathbf{X}}$ *linear GCN* $\xrightarrow{\text{further condition on } \mathbf{y}_L}$ *residual propagation*

$E[\mathbf{y}_U|\mathbf{X}] = [(\mathbf{I}+\omega\mathbf{N})^{-1}\mathbf{X}\beta]_U \qquad E[\mathbf{y}_U|\mathbf{X},\mathbf{y}_L] = \cdots$

Statistical Model for Node Attributes

Data Type | Corresponding Gaussian MRF | Learning Algorithm

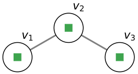
i.i.d. data (**X, y**)



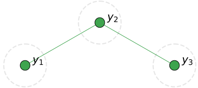
condition on **X** → *e.g. L={1,3}, U={2}*

linear regression

$E[\mathbf{y}_U|\mathbf{X}] = \mathbf{X}_U\beta \qquad \beta = (\mathbf{X}_L^\top\mathbf{X}_L)^{-1}\mathbf{X}_L^\top\mathbf{y}_L$



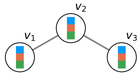
graph data (**y**, G)



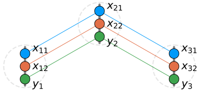
condition on $\mathbf{y}_L$ →

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 (**X**, **y**, G)



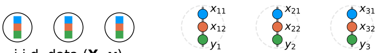
condition on **X** →

linear GCN $\xrightarrow{\text{further condition on } \mathbf{y}_L}$ *residual propagation*

$E[\mathbf{y}_U|\mathbf{X}] = [(\mathbf{I}+\omega\mathbf{N})^{-1}\mathbf{X}\beta]_U \qquad E[\mathbf{y}_U|\mathbf{X},\mathbf{y}_L] = \cdots$

Statistical Model for Node Attributes

Data Type | Corresponding Gaussian MRF | Learning Algorithm



i.i.d. data ($\mathbf{X}$, $\mathbf{y}$)

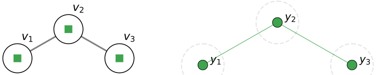
condition on $\mathbf{X}$ (*e.g.* $L=\{1,3\}$, $U=\{2\}$) $\longrightarrow$

linear regression

$E[\mathbf{y}_U|\mathbf{X}] = \mathbf{X}_U\beta \qquad \beta = (\mathbf{X}_L^\top\mathbf{X}_L)^{-1}\mathbf{X}_L^\top\mathbf{y}_L$

$\downarrow$ +nonlinearity

MLP

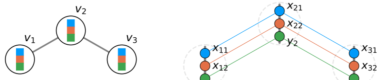


graph data ($\mathbf{y}$, G)

condition on $\mathbf{y}_L$ $\longrightarrow$

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)

condition on $\mathbf{X}$ $\longrightarrow$

linear GCN $\xrightarrow{\text{further condition on } \mathbf{y}_L}$ *residual propagation*

$E[\mathbf{y}_U|\mathbf{X}] = [(\mathbf{I}+\omega\mathbf{N})^{-1}\mathbf{X}\beta]_U \qquad E[\mathbf{y}_U|\mathbf{X},\mathbf{y}_L] = \cdots$

$\downarrow$ change filter

SGC (simple graph convolution)

$\downarrow$ +nonlinearity

GCN (graph convolution network)

Statistical Model for Node Attributes

The joint distribution of all node attributes is a Gaussian MRF

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

Statistical Model for Node Attributes

The joint distribution of all node attributes is a Gaussian MRF

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

Statistical Model for Node Attributes

The joint distribution of all node attributes is a Gaussian MRF

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

Statistical Model for Node Attributes

The joint distribution of all node attributes is a Gaussian MRF

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

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

Statistical Model for Node Attributes

The joint distribution of all node attributes is a Gaussian MRF

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

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

Statistical Model for Node Attributes

The joint distribution of all node attributes is a Gaussian MRF

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

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

Statistical Model for Node Attributes

The joint distribution of all node attributes is a Gaussian MRF

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

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

Statistical Model for Node Attributes

The joint distribution of all node attributes is a Gaussian MRF

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

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

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

Statistical Model for Node Attributes

The joint distribution of all node attributes is a Gaussian MRF

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

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

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

Case 1. Linear Regression if no Edge Exists

This is a well-known result for standard linear models

- Gaussian MRF log-potential simplifies as:

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

Case 1. Linear Regression if no Edge Exists

This is a well-known result for standard linear models

- Gaussian MRF log-potential simplifies as:

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

Case 1. Linear Regression if no Edge Exists

This is a well-known result for standard linear models

- Gaussian MRF log-potential simplifies as:

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

- Probability density function can be factorized as:

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

Case 1. Linear Regression if no Edge Exists

This is a well-known result for standard linear models

- Gaussian MRF log-potential simplifies as:

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

- Probability density function can be factorized as:

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

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

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

Case 1. Linear Regression if no Edge Exists

This is a well-known result for standard linear models

- Gaussian MRF log-potential simplifies as:

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

- Probability density function can be factorized as:

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

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

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

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

Case 2. Label Propagation if no Feature

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

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

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

Case 2. Label Propagation if no Feature

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

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

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

Case 2. Label Propagation if no Feature

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

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

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

Case 2. Label Propagation if no Feature

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

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

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 $\boldsymbol{H}, \boldsymbol{h}$ reduce to positive scalars:

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

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 $\boldsymbol{H}, \boldsymbol{h}$ reduce to positive scalars:

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

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 $\boldsymbol{H}, \boldsymbol{h}$ reduce to positive scalars:

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

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**:

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

$\omega = h/H$;

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

Case 2. Label Propagation if no Feature

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

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

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**:

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

$\omega = h/H$; 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 $\boldsymbol{H}, \boldsymbol{h}$ reduce to positive scalars:

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

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**:

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

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

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

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

Case 2. Label Propagation if no Feature

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

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

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**:

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

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

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

$\alpha = \omega/(1+\omega) \in (0,1)$ 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 $\boldsymbol{H}, \boldsymbol{h}$ reduce to positive scalars:

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

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**:

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

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

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

$\alpha = \omega/(1+\omega) \in (0,1)$ 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 $\boldsymbol{H}, \boldsymbol{h}$ reduce to positive scalars:

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

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**:

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

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

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

$\alpha = \omega/(1+\omega) \in (0,1)$ 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 $\mathbf{y}$ in the full model conditioned only on features

- The joint distribution of all attributes is given by:

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

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:

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

- Predict $\mathbf{y}$ 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:

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

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

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

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:

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

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

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

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:

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

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

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

$(\boldsymbol{I}_n + \omega\boldsymbol{N})^{-1}\boldsymbol{X}$ 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:

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

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

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

$$= \underbrace{(\boldsymbol{I}_n + \omega\boldsymbol{N})^{-1}\,\boldsymbol{X}}_{\text{LP on features!}}\beta$$

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

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

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:

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

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

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

$$= \underbrace{(\boldsymbol{I}_n + \omega\boldsymbol{N})^{-1} \boldsymbol{X}}_{\text{LP on features!}}\beta$$

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

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

- 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:

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

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

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

$$= \underbrace{(\boldsymbol{I}_n + \omega\boldsymbol{N})^{-1}\,\boldsymbol{X}}_{\text{LP on features!}}\beta$$

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

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

- 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:

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

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

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

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

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

- 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

LGC [Jia-Benson 21]	$(1-\alpha)(\boldsymbol{I} + \alpha \boldsymbol{S} + \alpha^2 \boldsymbol{S}^2 + \ldots)\boldsymbol{X}\beta$	$\boldsymbol{S} = \boldsymbol{D}^{-1/2}\boldsymbol{W}\boldsymbol{D}^{-1/2}$
SGC [Wu+ 19]	$\tilde{\boldsymbol{S}}^K \boldsymbol{X}\beta$	$\tilde{\boldsymbol{S}} = (\boldsymbol{D}+\boldsymbol{I})^{-1/2}(\boldsymbol{W}+\boldsymbol{I})(\boldsymbol{D}+\boldsymbol{I})^{-1/2}$
GCN [Kipf-Welling 17]	$\sigma(\tilde{\boldsymbol{S}} \ldots \sigma(\tilde{\boldsymbol{S}}\boldsymbol{X}\boldsymbol{\Theta}^{(1)}) \ldots \boldsymbol{\Theta}^{(K)})\beta$	

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

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

Case 3. Linear Graph Convolution

LGC [Jia-Benson 21]	$(1-\alpha)(\boldsymbol{I}+\alpha\boldsymbol{S}+\alpha^2\boldsymbol{S}^2+\ldots)\boldsymbol{X}\beta$	$\boldsymbol{S}=\boldsymbol{D}^{-1/2}\boldsymbol{W}\boldsymbol{D}^{-1/2}$
SGC [Wu+ 19]	$\tilde{\boldsymbol{S}}^K\boldsymbol{X}\beta$	$\tilde{\boldsymbol{S}}=(\boldsymbol{D}+\boldsymbol{I})^{-1/2}(\boldsymbol{W}+\boldsymbol{I})(\boldsymbol{D}+\boldsymbol{I})^{-1/2}$
GCN [Kipf-Welling 17]	$\sigma(\tilde{\boldsymbol{S}}\ldots\sigma(\tilde{\boldsymbol{S}}\boldsymbol{X}\boldsymbol{\Theta}^{(1)})\ldots\boldsymbol{\Theta}^{(K)})\beta$	

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

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

Case 3. Linear Graph Convolution

LGC [Jia-Benson 21]	$(1-\alpha)(\boldsymbol{I}+\alpha\boldsymbol{S}+\alpha^2\boldsymbol{S}^2+\ldots)\boldsymbol{X}\beta$	$\boldsymbol{S}=\boldsymbol{D}^{-1/2}\boldsymbol{W}\boldsymbol{D}^{-1/2}$
SGC [Wu+ 19]	$\tilde{\boldsymbol{S}}^K\boldsymbol{X}\beta$	$\tilde{\boldsymbol{S}}=(\boldsymbol{D}+\boldsymbol{I})^{-1/2}(\boldsymbol{W}+\boldsymbol{I})(\boldsymbol{D}+\boldsymbol{I})^{-1/2}$
GCN [Kipf-Welling 17]	$\sigma(\tilde{\boldsymbol{S}}\ldots\sigma(\tilde{\boldsymbol{S}}\boldsymbol{X}\boldsymbol{\Theta}^{(1)})\ldots\boldsymbol{\Theta}^{(K)})\beta$	

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

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

Case 3. Linear Graph Convolution

LGC [Jia-Benson 21]	$(1-\alpha)(\boldsymbol{I} + \alpha \boldsymbol{S} + \alpha^2 \boldsymbol{S}^2 + \ldots)\boldsymbol{X}\beta$	$\boldsymbol{S} = \boldsymbol{D}^{-1/2}\boldsymbol{W}\boldsymbol{D}^{-1/2}$
SGC [Wu+ 19]	$\tilde{\boldsymbol{S}}^K \boldsymbol{X}\beta$	$\tilde{\boldsymbol{S}} = (\boldsymbol{D}+\boldsymbol{I})^{-1/2}(\boldsymbol{W}+\boldsymbol{I})(\boldsymbol{D}+\boldsymbol{I})^{-1/2}$
GCN [Kipf-Welling 17]	$\sigma(\tilde{\boldsymbol{S}} \ldots \sigma(\tilde{\boldsymbol{S}}\boldsymbol{X}\boldsymbol{\Theta}^{(1)}) \ldots \boldsymbol{\Theta}^{(K)})\beta$	

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

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

Case 3. Linear Graph Convolution

LGC [Jia-Benson 21]	$(1-\alpha)(\boldsymbol{I} + \alpha\boldsymbol{S} + \alpha^2\boldsymbol{S}^2 + \ldots)\boldsymbol{X}\beta$	$\boldsymbol{S} = \boldsymbol{D}^{-1/2}\boldsymbol{W}\boldsymbol{D}^{-1/2}$
SGC [Wu+ 19]	$\tilde{\boldsymbol{S}}^K\boldsymbol{X}\beta$	$\tilde{\boldsymbol{S}} = (\boldsymbol{D}+\boldsymbol{I})^{-1/2}(\boldsymbol{W}+\boldsymbol{I})(\boldsymbol{D}+\boldsymbol{I})^{-1/2}$
GCN [Kipf-Welling 17]	$\sigma(\tilde{\boldsymbol{S}}\ldots\sigma(\tilde{\boldsymbol{S}}\boldsymbol{X}\boldsymbol{\Theta}^{(1)})\ldots\boldsymbol{\Theta}^{(K)})\beta$	

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

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

Case 3. Linear Graph Convolution

LGC [Jia-Benson 21]	$(1-\alpha)(\boldsymbol{I} + \alpha \boldsymbol{S} + \alpha^2 \boldsymbol{S}^2 + \ldots)\boldsymbol{X}\beta$	$\boldsymbol{S} = \boldsymbol{D}^{-1/2}\boldsymbol{W}\boldsymbol{D}^{-1/2}$
SGC [Wu+ 19]	$\tilde{\boldsymbol{S}}^K \boldsymbol{X}\beta$	$\tilde{\boldsymbol{S}} = (\boldsymbol{D}+\boldsymbol{I})^{-1/2}(\boldsymbol{W}+\boldsymbol{I})(\boldsymbol{D}+\boldsymbol{I})^{-1/2}$
GCN [Kipf-Welling 17]	$\sigma(\tilde{\boldsymbol{S}} \ldots \sigma(\tilde{\boldsymbol{S}}\boldsymbol{X}\boldsymbol{\Theta}^{(1)}) \ldots \boldsymbol{\Theta}^{(K)})\beta$	

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

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

Case 3. Linear Graph Convolution

LGC [Jia-Benson 21]	$(1-\alpha)(\boldsymbol{I}+\alpha\boldsymbol{S}+\alpha^2\boldsymbol{S}^2+\dots)\boldsymbol{X}\beta$	$\boldsymbol{S}=\boldsymbol{D}^{-1/2}\boldsymbol{W}\boldsymbol{D}^{-1/2}$
SGC [Wu+ 19]	$\tilde{\boldsymbol{S}}^K\boldsymbol{X}\beta$	$\tilde{\boldsymbol{S}}=(\boldsymbol{D}+\boldsymbol{I})^{-1/2}(\boldsymbol{W}+\boldsymbol{I})(\boldsymbol{D}+\boldsymbol{I})^{-1/2}$
GCN [Kipf-Welling 17]	$\sigma(\tilde{\boldsymbol{S}}\dots\sigma(\tilde{\boldsymbol{S}}\boldsymbol{X}\boldsymbol{\Theta}^{(1)})\dots\boldsymbol{\Theta}^{(K)})\beta$	

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

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

Case 3. Linear Graph Convolution

LGC [Jia-Benson 21]	$(1-\alpha)(\boldsymbol{I}+\alpha\boldsymbol{S}+\alpha^2\boldsymbol{S}^2+\ldots)\boldsymbol{X}\beta$	$\boldsymbol{S}=\boldsymbol{D}^{-1/2}\boldsymbol{W}\boldsymbol{D}^{-1/2}$
SGC [Wu+ 19]	$\tilde{\boldsymbol{S}}^K\boldsymbol{X}\beta$	$\tilde{\boldsymbol{S}}=(\boldsymbol{D}+\boldsymbol{I})^{-1/2}(\boldsymbol{W}+\boldsymbol{I})(\boldsymbol{D}+\boldsymbol{I})^{-1/2}$
GCN [Kipf-Welling 17]	$\sigma(\tilde{\boldsymbol{S}}\ldots\sigma(\tilde{\boldsymbol{S}}\boldsymbol{X}\boldsymbol{\Theta}^{(1)})\ldots\boldsymbol{\Theta}^{(K)})\beta$	

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

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

Case 4. Residual Propagation

- The conditional distribution of all labels conditioned on features:

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

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

Case 4. Residual Propagation

- The conditional distribution of all labels conditioned on features:

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

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

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

Case 4. Residual Propagation

- The conditional distribution of all labels conditioned on features:

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

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

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

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

Case 4. Residual Propagation

- The conditional distribution of all labels conditioned on features:

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

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

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

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

Case 4. Residual Propagation

- The conditional distribution of all labels conditioned on features:

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

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

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

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

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

Case 4. Residual Propagation

- The conditional distribution of all labels conditioned on features:

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

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

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

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

- 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 $\boldsymbol{r}_L$ to testing residuals $\boldsymbol{r}_U$
  - add interpolated testing residuals $\boldsymbol{r}_U$ to initial predictions $\hat{\boldsymbol{y}}_U$ as corrections

Case 4. Residual Propagation

- The conditional distribution of all labels conditioned on features:

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

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

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

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

- 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 $\boldsymbol{r}_L$ to testing residuals $\boldsymbol{r}_U$
  - add interpolated testing residuals $\boldsymbol{r}_U$ to initial predictions $\hat{\boldsymbol{y}}_U$ 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

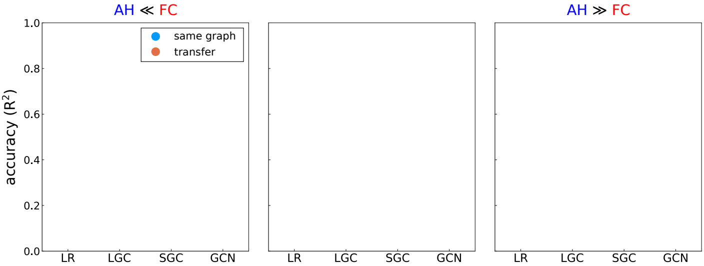
Our Model Connects a lot of Dots

	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

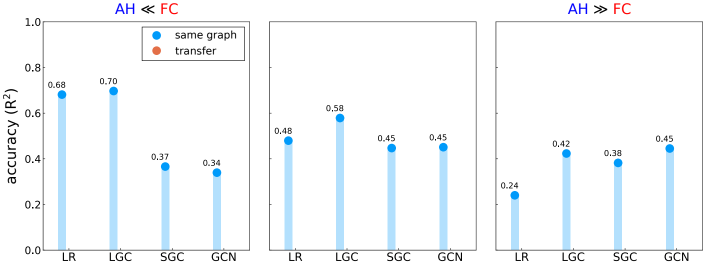


- same graph: sample one attributed graphs G_1, train and test on different vertices

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

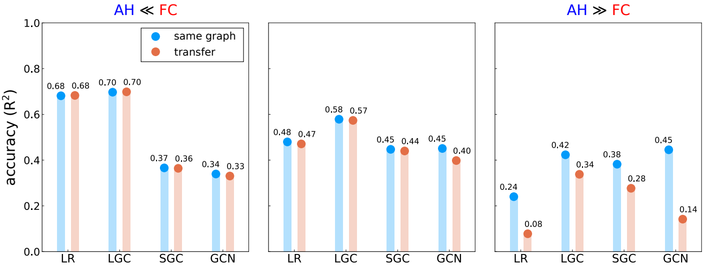


- same graph: sample one attributed graphs G_1, train and test on different vertices
- same graph: LGC is the best when FC is strong, GCN is better when AH $\gg$ FC **!?!**

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

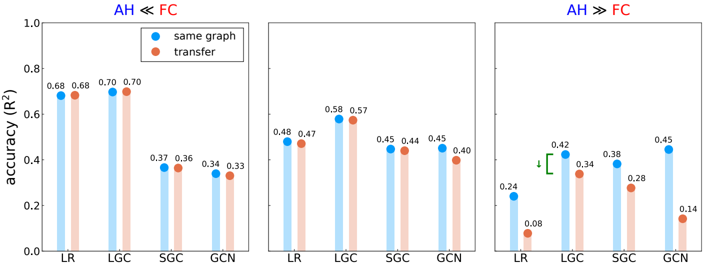


- same graph: sample one attributed graphs G_1, train and test on different vertices
  transfer: sample two attributed graphs G_1, G_2, train on G_1 and test on G_2
- same graph: LGC is the best when FC is strong, GCN is better when AH $\gg$ FC **!?!**

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

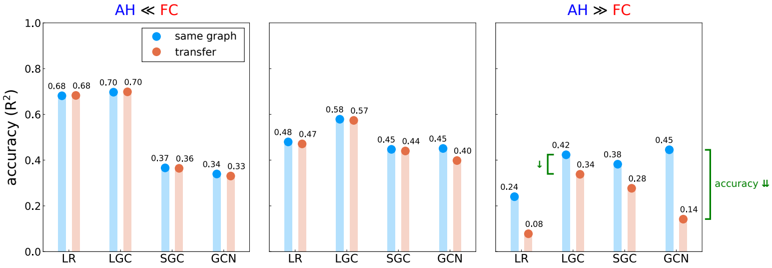


- same graph: sample one attributed graphs G_1, train and test on different vertices
  transfer: sample two attributed graphs G_1, G_2, train on G_1 and test on G_2
- same graph: LGC is the best when FC is strong, GCN is better when AH $\gg$ FC **!?!**
- performance degradation happens on transfer learning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

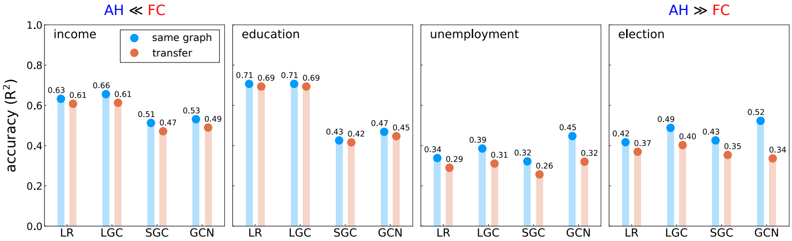


- same graph: sample one attributed graphs G_1, train and test on different vertices
  transfer: sample two attributed graphs G_1, G_2, train on G_1 and test on G_2
- same graph: LGC is the best when FC is strong, GCN is better when AH $\gg$ FC **!?!**
- performance degradation happens on transfer learning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

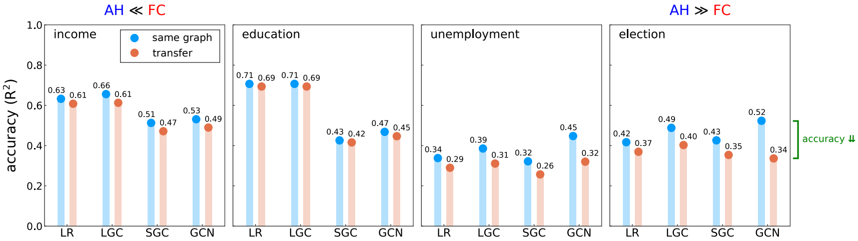


- same graph: sample one attributed graphs G_1, train and test on different vertices
  transfer: sample two attributed graphs G_1, G_2, train on G_1 and test on G_2
- same graph: LGC is the best when FC is strong, GCN is better when AH $\gg$ FC **!?!**
- performance degradation happens on transfer learning 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 $\boldsymbol{N} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^{\intercal}$.

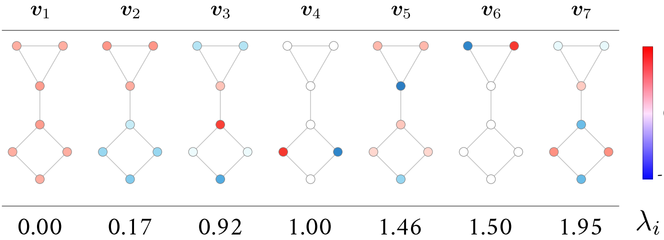
[Li+ 2018; Li+ 2019]

Our Model Helps Understand Smoothing

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

[Li+ 2018; Li+ 2019]

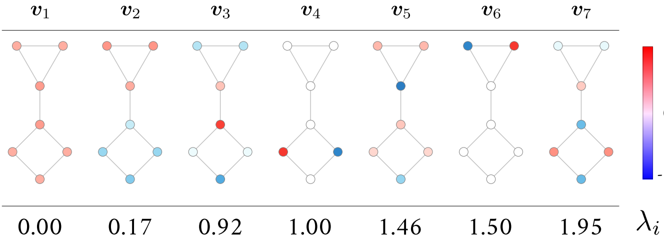
Our Model Helps Understand Smoothing



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

[Li+ 2018; Li+ 2019]

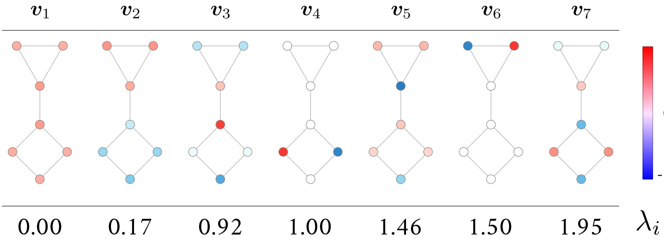
Our Model Helps Understand Smoothing



- Spectral graph theory measures smoothness with eigenvectors of $\boldsymbol{N} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^\intercal$. any graph signal can be decomposed in the eigenbasis $\boldsymbol{f} = \sum_i c_i \boldsymbol{v}_i$

[Li+ 2018; Li+ 2019]

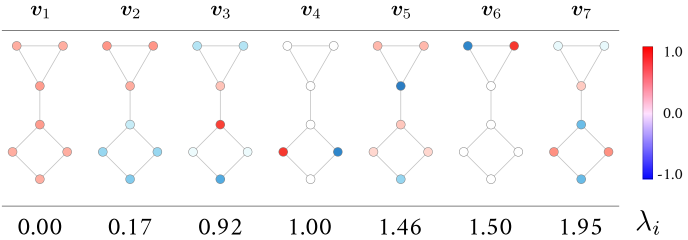
Our Model Helps Understand Smoothing



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

[Li+ 2018; Li+ 2019]

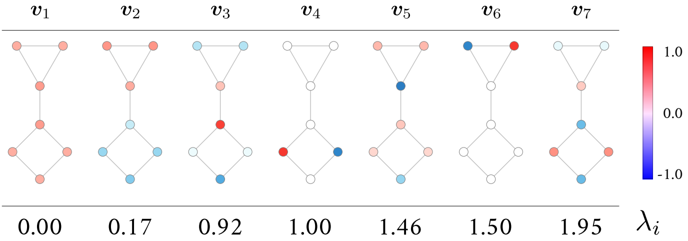
Our Model Helps Understand Smoothing



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

[Li+ 2018; Li+ 2019]

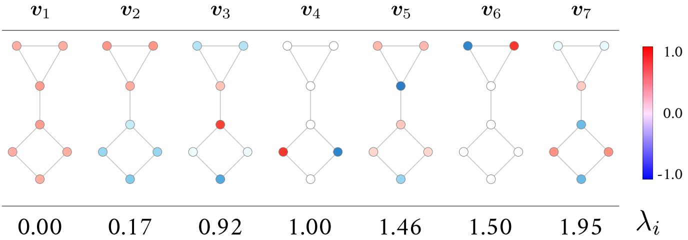
Our Model Helps Understand Smoothing



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

[Li+ 2018; Li+ 2019]

Our Model Helps Understand Smoothing

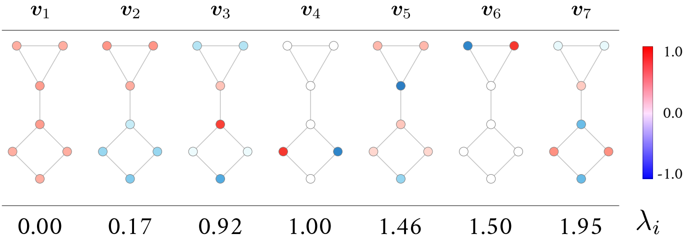


- Spectral graph theory measures smoothness with eigenvectors of $\boldsymbol{N} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^\intercal$.
  any graph signal can be decomposed in the eigenbasis $\boldsymbol{f} = \sum_i c_i \boldsymbol{v}_i$
- Why feature smoothing helps prediction? How are LGC and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

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

[Li+ 2018; Li+ 2019]

Our Model Helps Understand Smoothing



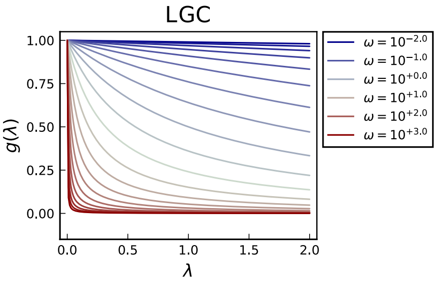
- Spectral graph theory measures smoothness with eigenvectors of $\boldsymbol{N} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^\intercal$.
  any graph signal can be decomposed in the eigenbasis $\boldsymbol{f} = \sum_i c_i \boldsymbol{v}_i$
- Why feature smoothing helps prediction? How are LGC and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

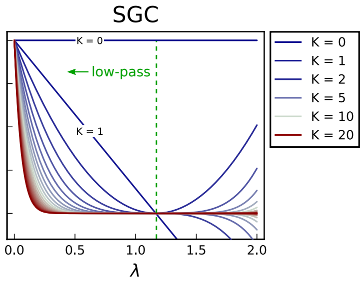
LGC :
$$(\boldsymbol{I} + \omega \boldsymbol{N})^{-1} \boldsymbol{f} = \sum_i c_i \boldsymbol{v}_i \ \ (1 + \omega \lambda_i)^{-1}$$

SGC :
$$\tilde{\boldsymbol{S}}^K \boldsymbol{f} = \sum_i c_i \boldsymbol{v}_i \ \ (1 - {}^{d}\!/\!_{(d+1)} \lambda_i)^K$$

[Li+ 2018; Li+ 2019]

Our Model Helps Understand Smoothing

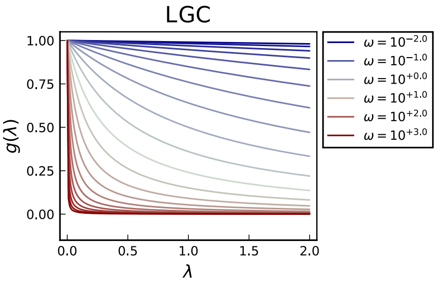


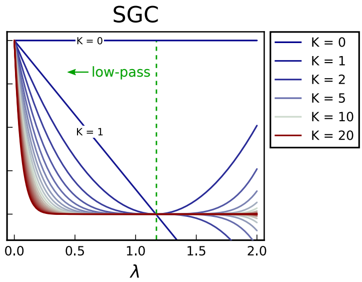


$$(\boldsymbol{I}+\omega\boldsymbol{N})^{-1}\boldsymbol{f} = \sum_i c_i \boldsymbol{v}_i \ \ (1+\omega\lambda_i)^{-1}$$

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

Our Model Helps Understand Smoothing

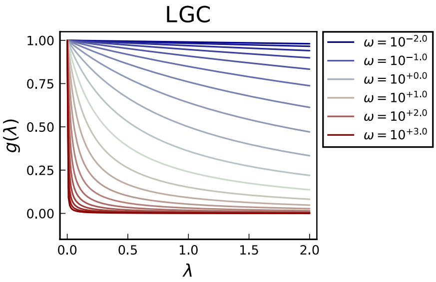


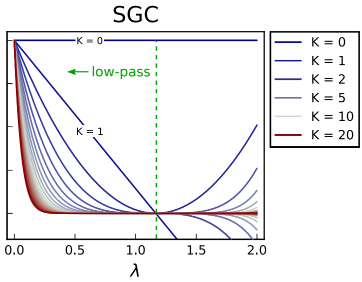


$$(\boldsymbol{I}+\omega \boldsymbol{N})^{-1}\boldsymbol{f} = \sum_i c_i \boldsymbol{v}_i \;\; (1+\omega\lambda_i)^{-1} \qquad\qquad \tilde{\boldsymbol{S}}^K \boldsymbol{f} = \sum_i c_i \boldsymbol{v}_i \;\; (1 - {}^{d}\!/\!{}_{(d+1)}\lambda_i)^K$$

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

Our Model Helps Understand Smoothing

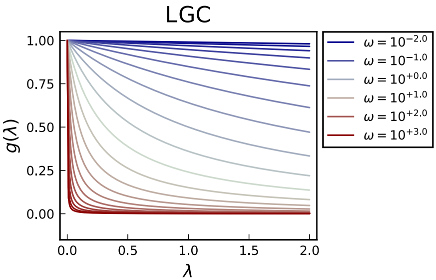


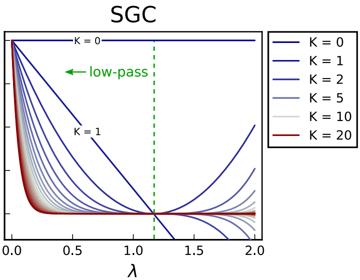


$$(\boldsymbol{I}+\omega\boldsymbol{N})^{-1}\boldsymbol{f} = \sum_i c_i\boldsymbol{v}_i \;\; (1+\omega\lambda_i)^{-1} \qquad \tilde{\boldsymbol{S}}^K\boldsymbol{f} = \sum_i c_i\boldsymbol{v}_i \;\; (1-{}^{d}/_{(d+1)}\lambda_i)^K$$

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

Our Model Helps Understand Smoothing



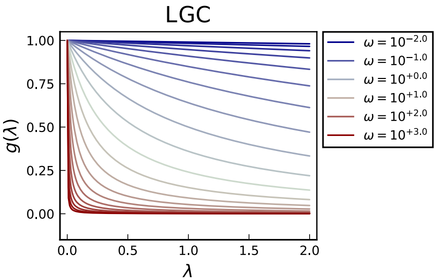


$$(\boldsymbol{I}+\omega\boldsymbol{N})^{-1}\boldsymbol{f} = \sum_i c_i \boldsymbol{v}_i \ \ (1+\omega\lambda_i)^{-1} \qquad \tilde{\boldsymbol{S}}^K\boldsymbol{f} = \sum_i c_i \boldsymbol{v}_i \ \ (1 - {}^{d}/_{(d+1)}\lambda_i)^K$$

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

  better balance between perserving the signal $(\omega, K\downarrow)$ and reducing the noise $(\omega, K\uparrow)$.

Our Model Helps Understand Smoothing

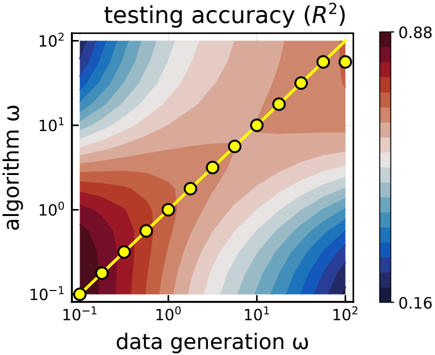


$$(\boldsymbol{I}+\omega\boldsymbol{N})^{-1}\boldsymbol{f} = \sum_i c_i \boldsymbol{v}_i \ \ (1+\omega\lambda_i)^{-1} \qquad \tilde{\boldsymbol{S}}^K \boldsymbol{f} = \sum_i c_i \boldsymbol{v}_i \ \ (1 - {}^{d}/_{(d+1)}\lambda_i)^K$$

- LGC is a low-pass filter on the entire eigenspectrum $[0, 2]$
- LGC is more flexible than SGC, due to its continuous parameter ω

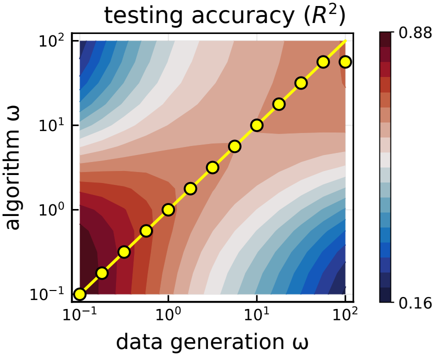
better balance between perserving the signal $(\omega, K \downarrow)$ and reducing the noise $(\omega, K \uparrow)$.

Our Model Helps Understand Smoothing



- Sample datasets from our generative model, with different $\omega = h/H$

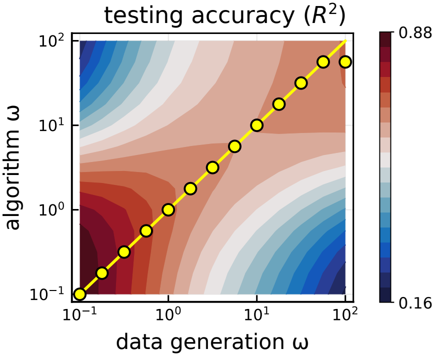
Our Model Helps Understand Smoothing



- Sample datasets from our generative model, with different $\omega = h/H$

  For each dataset, run algo with different ω; create a 2D performance plot

Our Model Helps Understand Smoothing

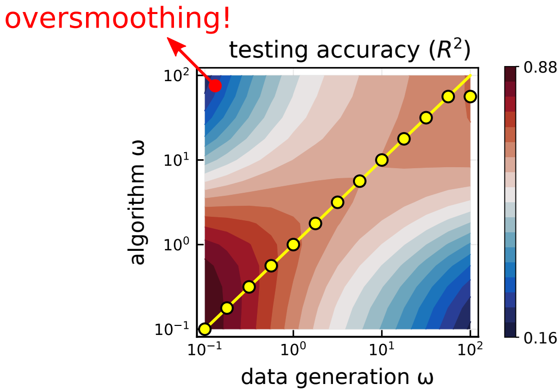


- Sample datasets from our generative model, with different $\omega = h/H$

  For each dataset, run algo with different ω; create a 2D performance plot

- Performance the best along the diagonal (algo $\omega =$ data ω)

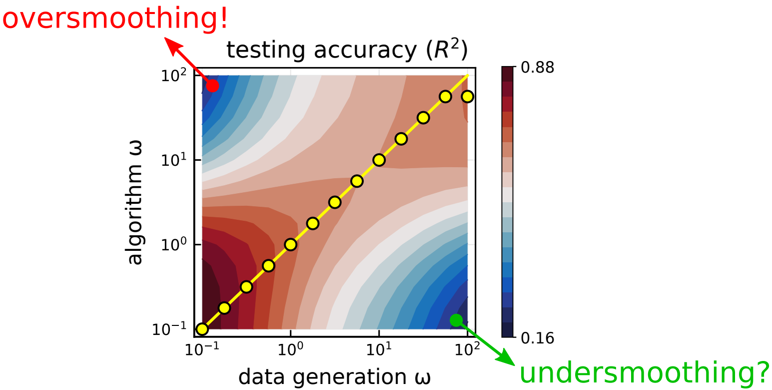
Our Model Helps Understand Smoothing



- 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 $\omega =$ data ω)
- Oversmoothing happens if data ω is small, but algorithm ω is large

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 $\omega =$ data ω)
- 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 negiative)
- 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 $\rightarrow$ older population
  - lower unemployment rate, and higher migration rate $\rightarrow$ manufacturing hubs?
  - lower sh100m, and higher sh500m $\rightarrow$ rural areas?

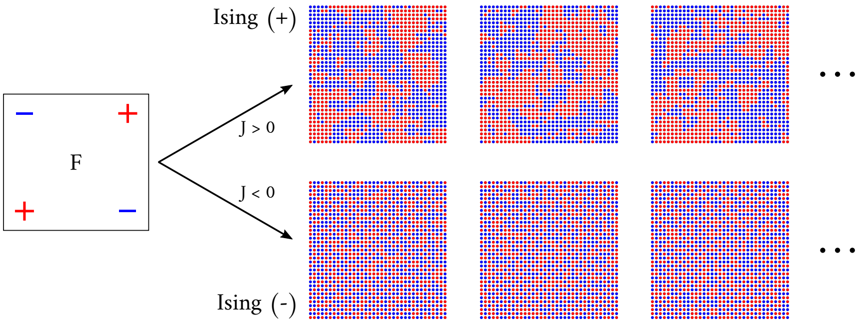
sh050m/sh100m/sh500m: shares of friends within 50/100/500 miles; features normalized to zero mean & unit variance

Where do we go from here?

- What if the labels are not positively correlated?
  (e.g. heterophily graphs)

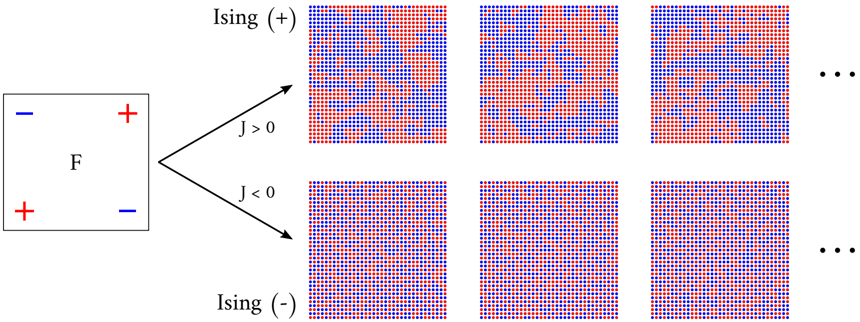
- What if the labels are categorical variables?
  (node classification setting)

What if labels aren't Poistively Correlated?



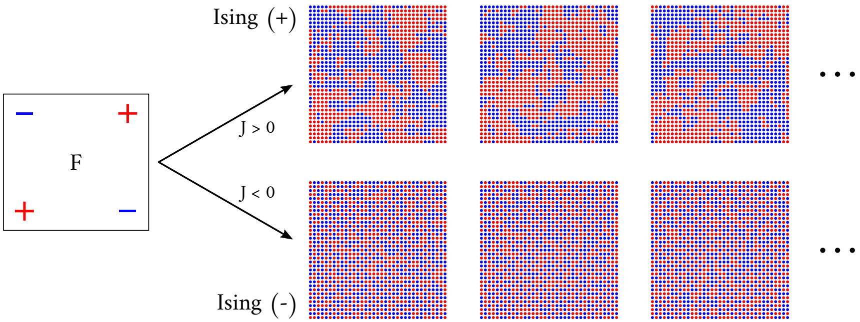
- Sometimes labels can be negatively correlated among neigbors.

What if labels aren't Poistively Correlated?



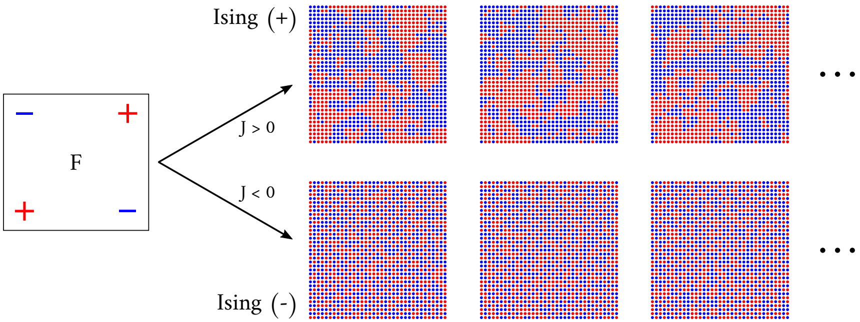
- Sometimes labels can be negatively correlated among neigbors.
  - Ising model with negative coupling J (antiferromagnetic materials)

What if labels aren't Poistively Correlated?



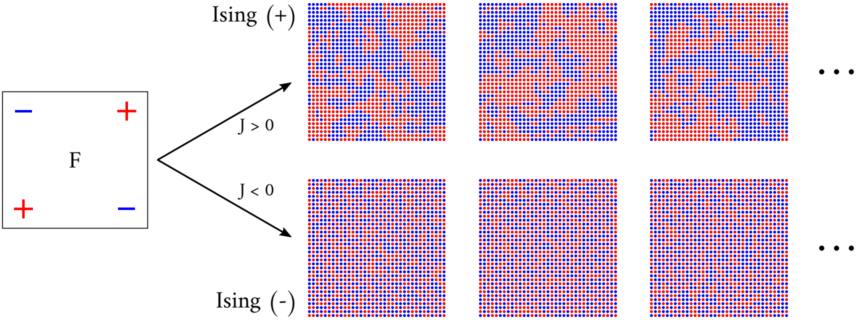
- Sometimes labels can be negatively correlated among neigbors.
  - Ising model with negative coupling J (antiferromagnetic materials)

What if labels aren't Poistively Correlated?



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

What if labels aren't Poistively Correlated?



- Sometimes labels can be negatively correlated among neigbors.
  - 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:

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

We can learn the Correlation Directly

- Model the residual with a multivariate Gaussian:

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

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

We can learn the Correlation Directly

- Model the residual with a multivariate Gaussian:

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

  - $\beta > 0$ 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:

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

  - $\beta > 0$ 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

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

We can learn the Correlation Directly

- Model the residual with a multivariate Gaussian:

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

  - $\beta > 0$ 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

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

- How do we determine the parameter α?

We can learn the Correlation Directly

- Model the residual with a multivariate Gaussian:

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

  - $\beta > 0$ 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

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

- 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 $\boldsymbol{r}_L$

[Ubaru+ 2017; Gardner+ 2018]

We can learn the Correlation Directly

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

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

[Ubaru+ 2017; Gardner+ 2018]

We can learn the Correlation Directly

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

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

[Ubaru+ 2017; Gardner+ 2018]

We can learn the Correlation Directly

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

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

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

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

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

[Ubaru+ 2017; Gardner+ 2018]

We can learn the Correlation Directly

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

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

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

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

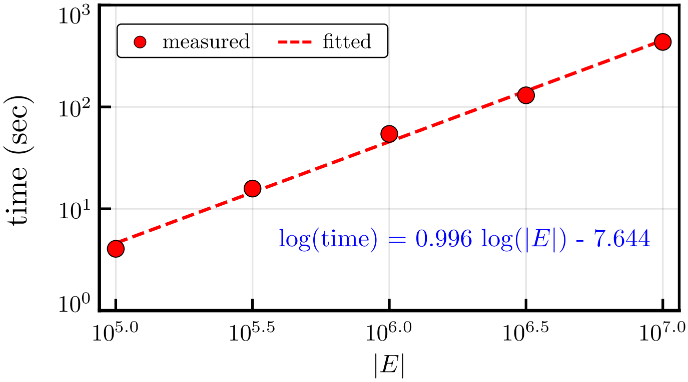
  - $\boldsymbol{z}_t$ random Gaussian trial vectors with unit-variance
  - ω_{ti}/ξ_{ti} numerical integration weights/nodes computed with Lanczos algorithm

- Estimate gradient with stochastic trace estimator & CG solver

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

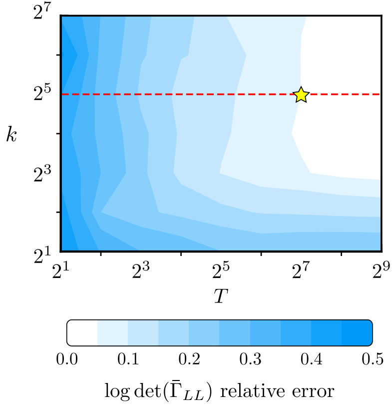
[Ubaru+ 2017; Gardner+ 2018]

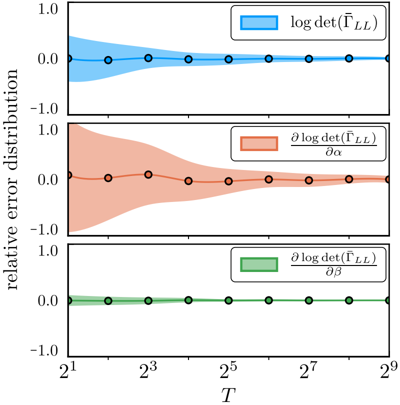
Stochastic Estimation is Efficient



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

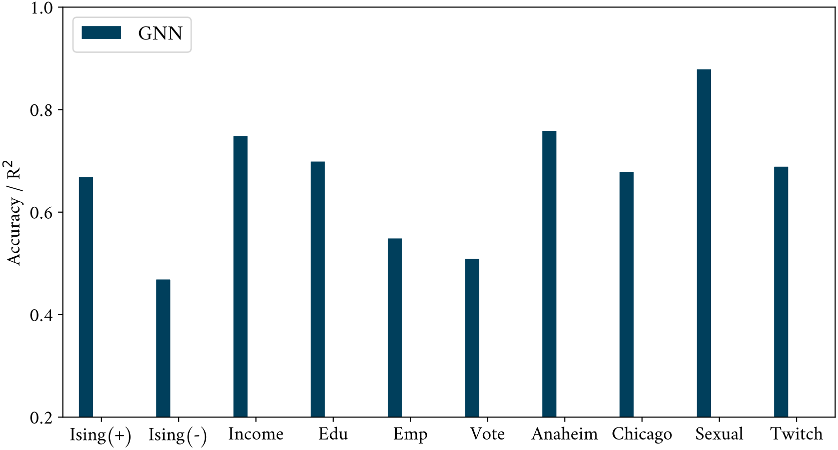
Stochastic Estimation is Accurate



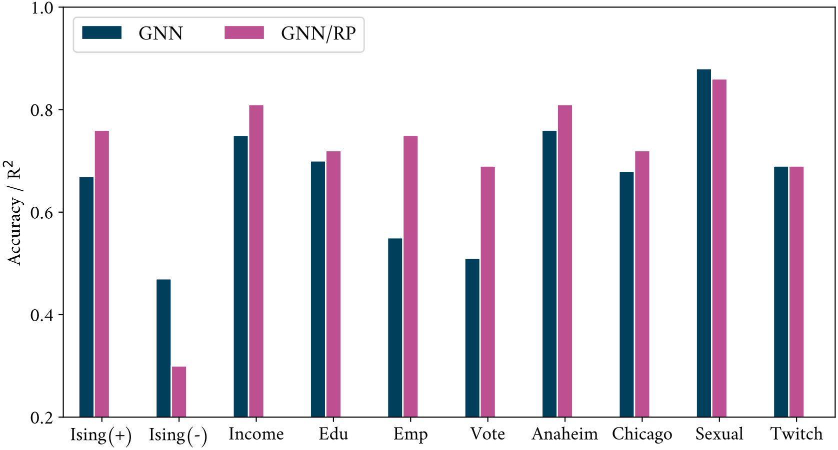


- Stochastic trace estimator gives accurate results
  - a small number of CG iterations, Gaussian trial vectors; $< 5\%$ error
  - unbiased estimation of gradient optimize with SGD

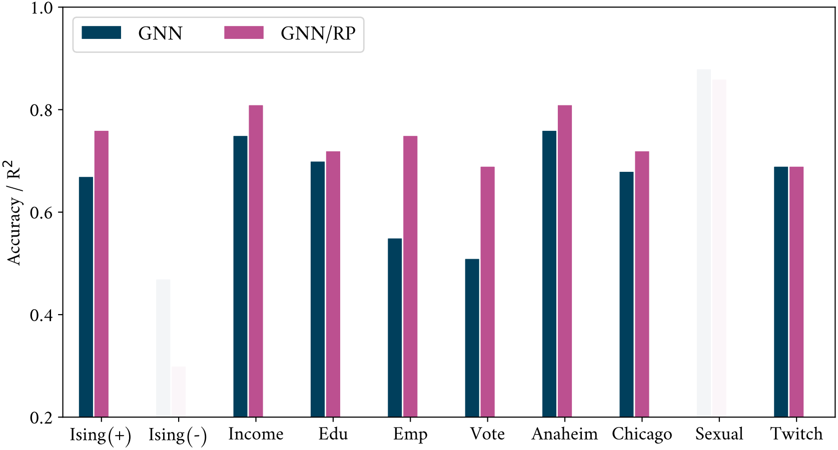
C-GNN Performs well on Heterophily Graphs



C-GNN Performs well on Heterophily Graphs

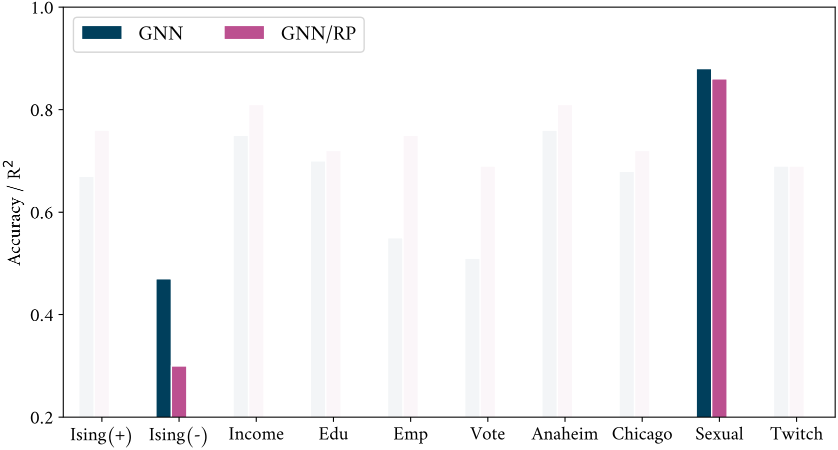


C-GNN Performs well on Heterophily Graphs



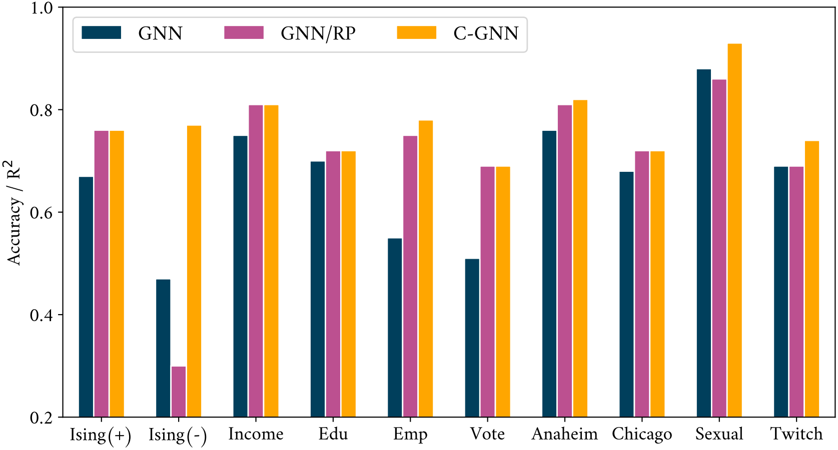
- GNN/RP outperforms GNN on homophily graphs;

C-GNN Performs well on Heterophily Graphs



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

C-GNN Performs well on Heterophily 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:

$\boldsymbol{h} \in \mathbb{R}^c$, $\boldsymbol{H} \in \mathbb{R}^{c \times c}$ 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

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

$\boldsymbol{h} \in \mathbb{R}^c$, $\boldsymbol{H} \in \mathbb{R}^{c \times c}$ 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

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

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

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

$\boldsymbol{h} \in \mathbb{R}^c$, $\boldsymbol{H} \in \mathbb{R}^{c \times c}$ 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

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

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

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

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

$\boldsymbol{h} \in \mathbb{R}^c$, $\boldsymbol{H} \in \mathbb{R}^{c \times c}$ 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

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

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

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

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

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

$\boldsymbol{h} \in \mathbb{R}^c$, $\boldsymbol{H} \in \mathbb{R}^{c \times c}$ 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

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

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

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

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

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

which is just another pairwise MRF.

$\boldsymbol{h} \in \mathbb{R}^c$, $\boldsymbol{H} \in \mathbb{R}^{c \times c}$ 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

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

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

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

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

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

  which is just another pairwise MRF.

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

Learning algo from Probabilistic Inference

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

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

Learning algo from Probabilistic Inference

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

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

Learning algo from Probabilistic Inference

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

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

Learning algo from Probabilistic Inference

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

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

Learning algo from Probabilistic Inference

$$P(\boldsymbol{y}|\boldsymbol{X}) \cong \prod_{(u,v)\in E} H(y_u, y_v) \prod_{u\in V} P(\boldsymbol{x}_u|y_u) \cdot h(y_u)$$
$$= \prod_{(u,v)\in E} {\color{red}H}(y_u, y_v) \prod_{u\in V} f_{\color{red}\theta}(y_u; \boldsymbol{x}_u)$$

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

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

Learning algo from Probabilistic Inference

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

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

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

  - 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

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

- Probabilistic inference: compute $P(y_u|\boldsymbol{X})$ with **belief propagation**
- Since $P(\boldsymbol{x}_u|y_u), \boldsymbol{h}, \boldsymbol{H}$ are unkonwn, 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

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

  - 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

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

  - 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 $P(y_u|\boldsymbol{X})$.

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

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

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

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

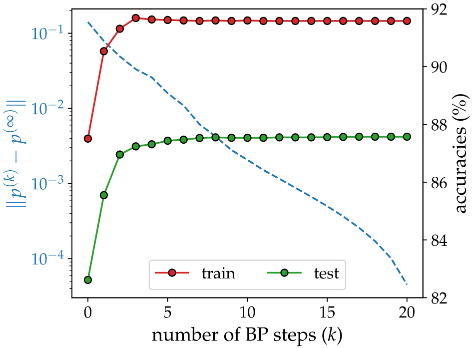
$$m_{vu}^{(k)}(y_u) \cong \sum_{y_v} H_{vu}(y_v, y_u) \cdot p_v^{(k-1)}(y_v) \cdot m_{uv}^{(k-1)}(y_v)^{-1}$$

- After training, we can infer unknown labels directly $P(y_u|\boldsymbol{X})$.
- Or we can compute $P(y_u|\boldsymbol{X}, \boldsymbol{y}_L)$:

  further conditioning on $\boldsymbol{y}_L$, resulting in a pairwise MRF on $G[U]$

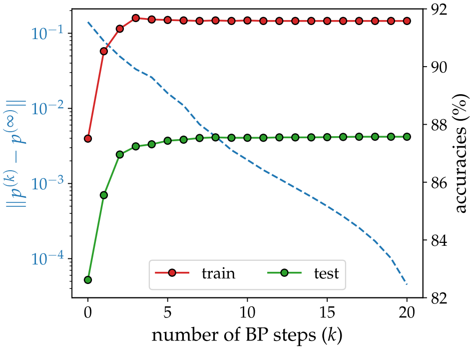
$$P(\boldsymbol{y}_U|\boldsymbol{X}, \boldsymbol{y}_L) \cong \prod_{u \in U} \left(f_\theta(y_u; \boldsymbol{x}_u) \prod_{\substack{u \in U, v \in L, \\ (u,v) \in E}} H_{uv}(y_u, y_v) \right) \prod_{\substack{u, v \in U, \\ (u,v) \in E}} H_{uv}(y_u, y_v)$$

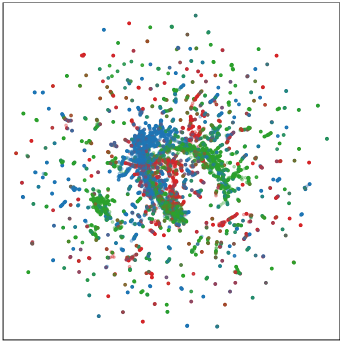
GBPN Case Study on PubMed Dataset



- Belief propagation converge quickly on real-world dataset

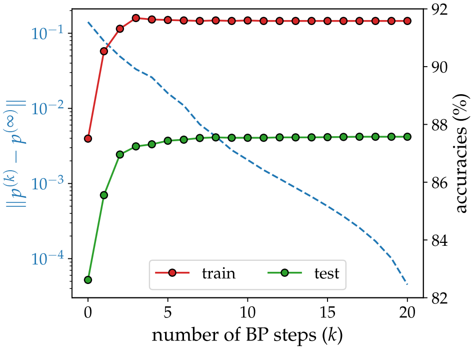
GBPN Case Study on PubMed Dataset

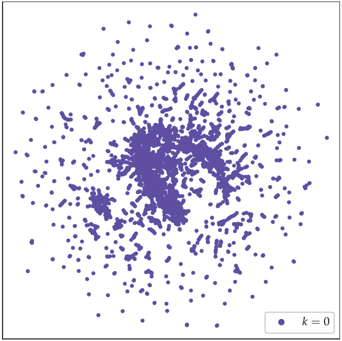




- Belief propagation converge quickly on real-world dataset
- Visualizing ground truth labels on PubMed

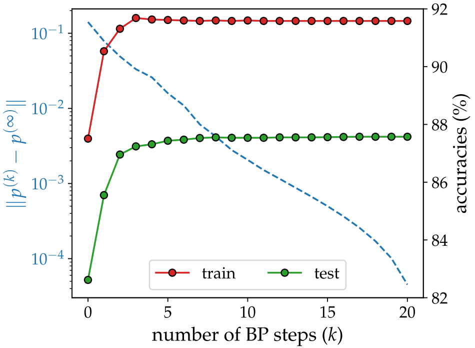
GBPN Case Study on PubMed Dataset

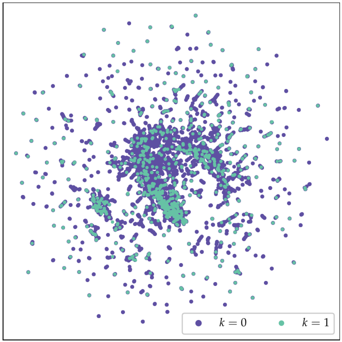




- Belief propagation converge quickly on real-world dataset
- Visualizing ground truth labels on PubMed
- GBPN is interpretable (node correctly predicted by GBPN after k steps)

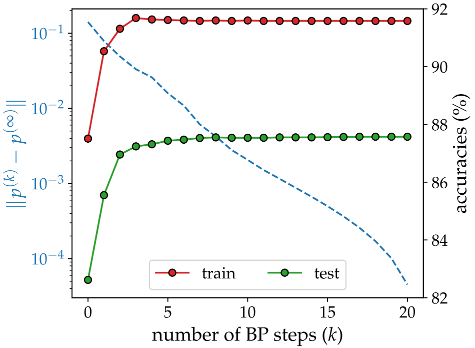
GBPN Case Study on PubMed Dataset

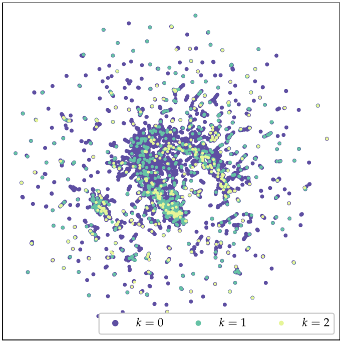




- Belief propagation converge quickly on real-world dataset
- Visualizing ground truth labels on PubMed
- GBPN is interpretable (node correctly predicted by GBPN after k steps)

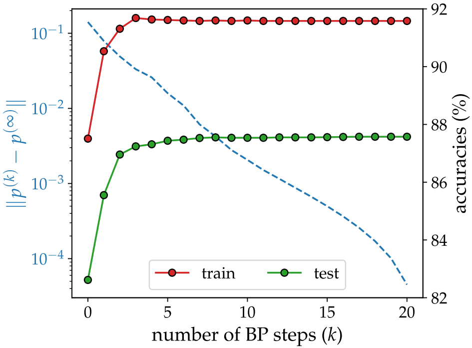
GBPN Case Study on PubMed Dataset

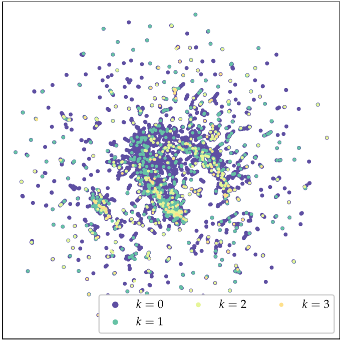




- Belief propagation converge quickly on real-world dataset
- Visualizing ground truth labels on PubMed
- GBPN is interpretable (node correctly predicted by GBPN after k steps)

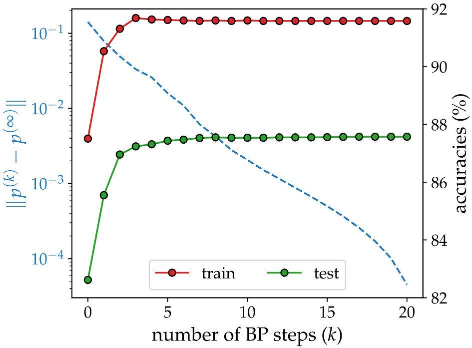
GBPN Case Study on PubMed Dataset

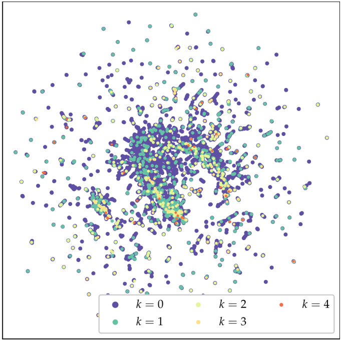




- Belief propagation converge quickly on real-world dataset
- Visualizing ground truth labels on PubMed
- GBPN is interpretable (node correctly predicted by GBPN after k steps)

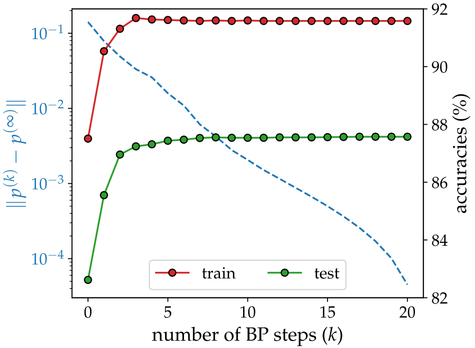
GBPN Case Study on PubMed Dataset

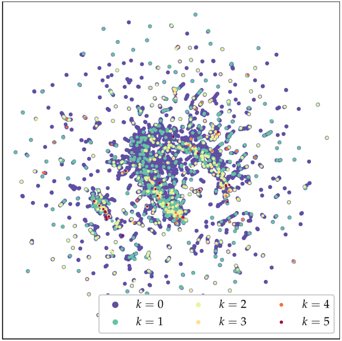




- Belief propagation converge quickly on real-world dataset
- Visualizing ground truth labels on PubMed
- GBPN is interpretable (node correctly predicted by GBPN after k steps)

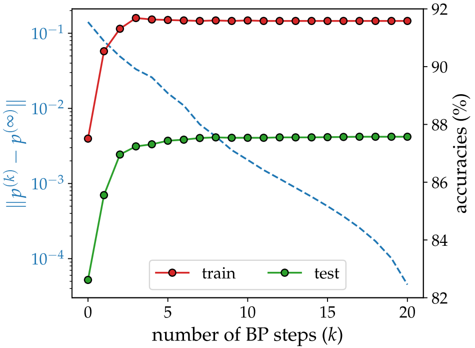
GBPN Case Study on PubMed Dataset

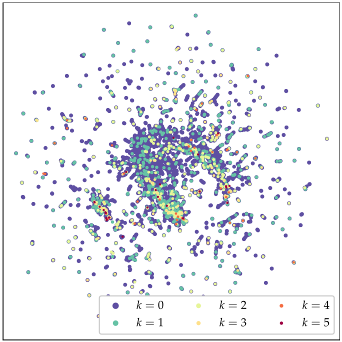




- Belief propagation converge quickly on real-world dataset
- Visualizing ground truth labels on PubMed
- GBPN is interpretable (node correctly predicted by GBPN after k steps)

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

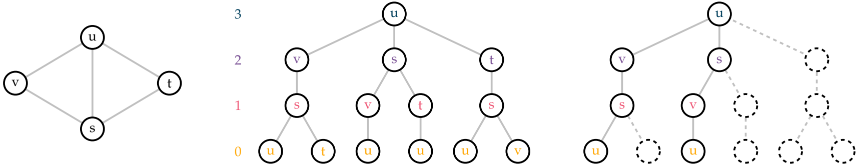
dataset	MLP	SGC	GCN	GraphSAGE	GAT	GBPN(I)	GBPN(T)
County	89.8 ± 0.6	88.2 ± 0.7	87.9 ± 0.7	**90.9** ± 0.5	90.5 ± 0.4	90.1 ± 0.7	90.4 ± 0.8
Sexual	73.9 ± 1.4	76.1 ± 1.4	83.7 ± 1.1	93.0 ± 0.9	93.7 ± 0.9	97.0 ± 0.5	**97.1** ± 0.5
Cora	72.4 ± 1.1	**87.1** ± 0.7	**87.1** ± 0.7	**87.1** ± 0.8	86.7 ± 0.7	84.8 ± 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

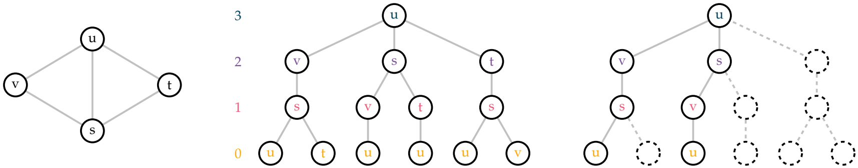
- GBPN is not only more interpretable, but also more accurate

Performance on Large-scale Datasets



- We design a subsampling algorithm for mini-batch training

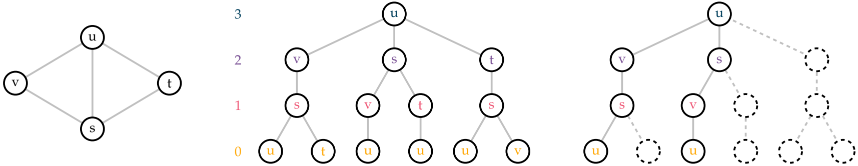
Performance on Large-scale Datasets



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

- We design a subsampling algorithm for mini-batch training

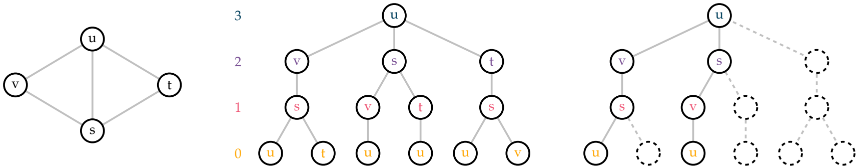
Performance on Large-scale Datasets



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 noticable margin on some dataset

Conclusions

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

Conclusions

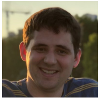
- One key aspect of semi-supervised node regression is to leverage feature correlation and attribute homophily.
- My thesis proposed a generative model that accounts for three different types of correlations in attributed graphs. It unifies existing algorithms and motivates new ones with the state-of-the-art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