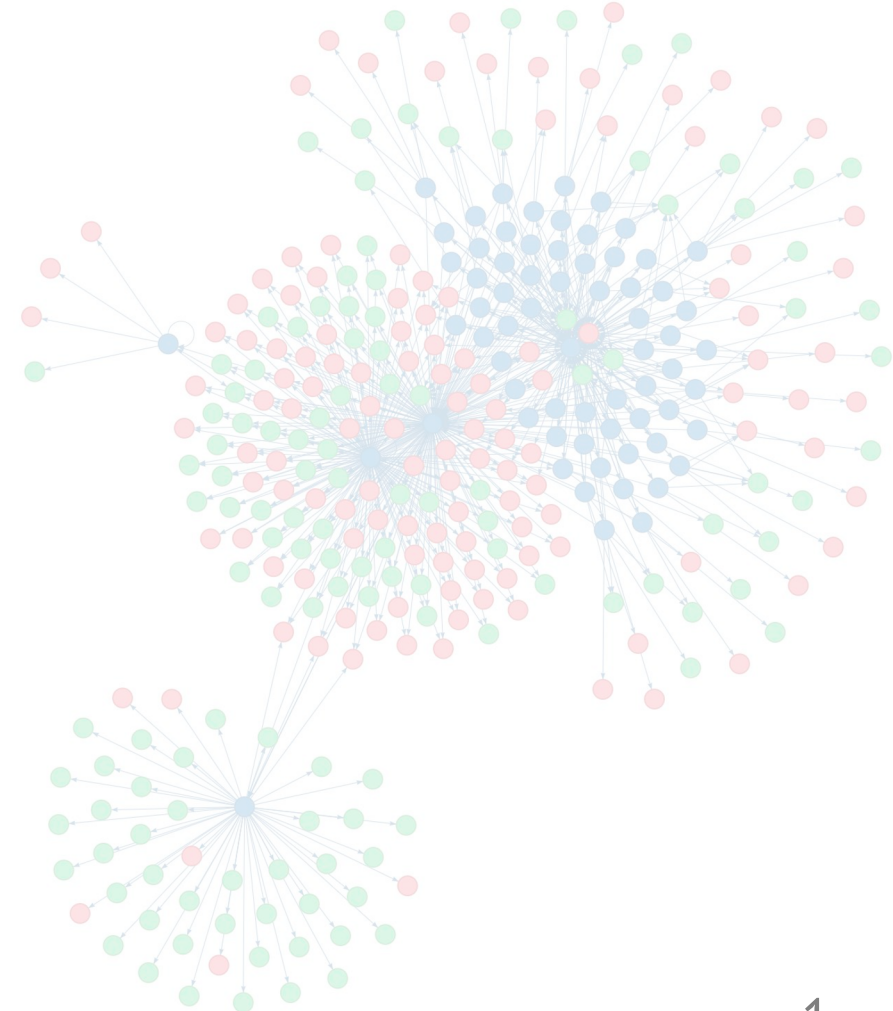


Label Propagation and Graph Neural Networks

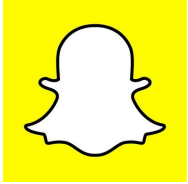
Austin R. Benson · Cornell University
RelationalAI · February 23, 2021



Joint work with
Junteng Jia (Cornell)



Graph data modeling complex systems are everywhere.



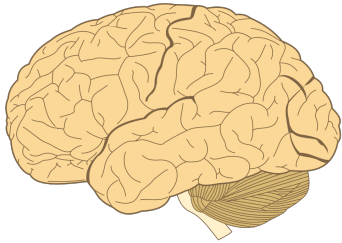
Society

nodes are people
edges are friendships



Finance

nodes are accounts
edges are transactions



Brains

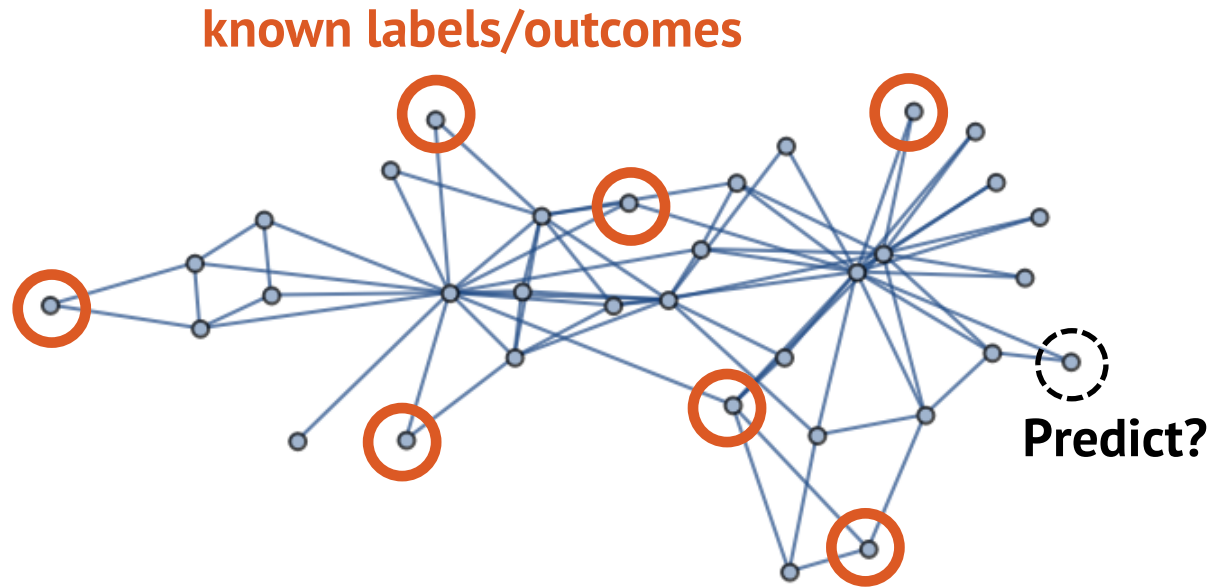
nodes are neurons
edges are synapses



Commerce

nodes are products
edges are copurchases

We often want to predict/estimate/construct/forecast attributes/labels/outcomes/clusters on nodes.



- Bad actors in financial transaction graphs [Weber+ 18, 19; Pareja+ 20]
 - Gender in social networks [Peel 17; Altenburger–Ugander 18]
 - Document classification in citation networks [Lu–Getoor 03; Kipf–Welling 17]
 - Product categories from coreview/copurchase [Huang+ 20; Veldt+ 20]
 - Election outcomes from social connections [Jia–Benson 21]
-
- Might have rich additional info on nodes (features)
transaction history, user interests, document text, product ratings, demographics
 - Graph-based semi-supervised learning, clustering, node prediction, relational learning, collective classification, community detection, ...

The formal problem we are solving.

Problem input.

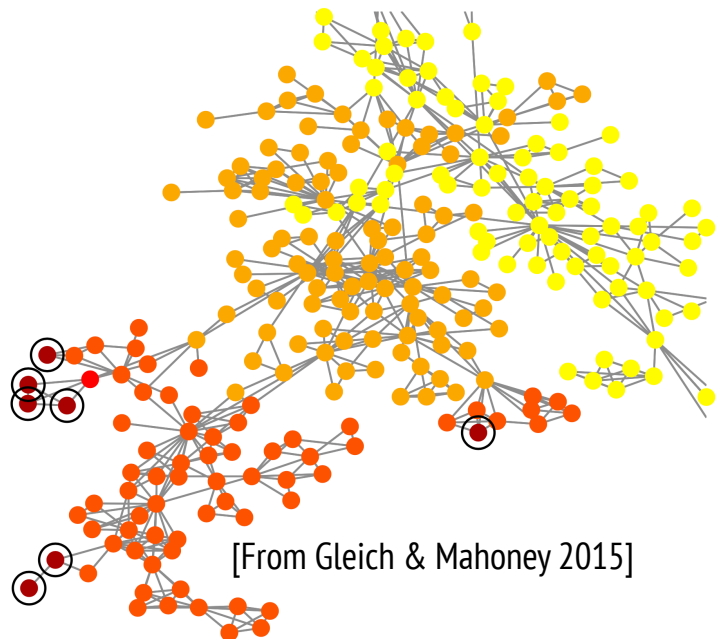
- Graph $G = (V, E)$.
- $|V| \times p$ matrix \mathbf{X} of node features.
- Subset $L \subset V$ of labeled nodes.
- Length- $|L|$ vector \mathbf{y}_L of outcomes on L (real-valued or categorical).

Problem output.

- Length- $|U|$ vector \mathbf{y}_U of outcomes on $U = V \setminus L$.

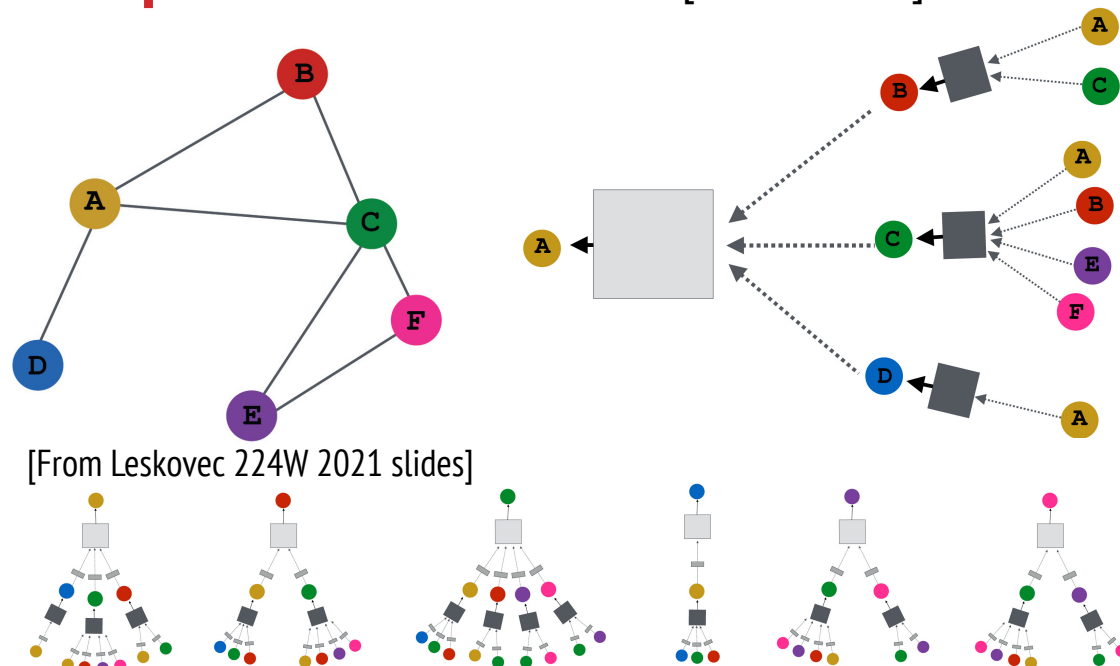
There are two broad classes of algorithms.

1. Label Propagation [early 2000s]



- Propagate/spread/diffuse known values.
- Classification: once per label \rightarrow scores.
- Regression: spread real values.

2. Graph Neural Networks [late 2010s]



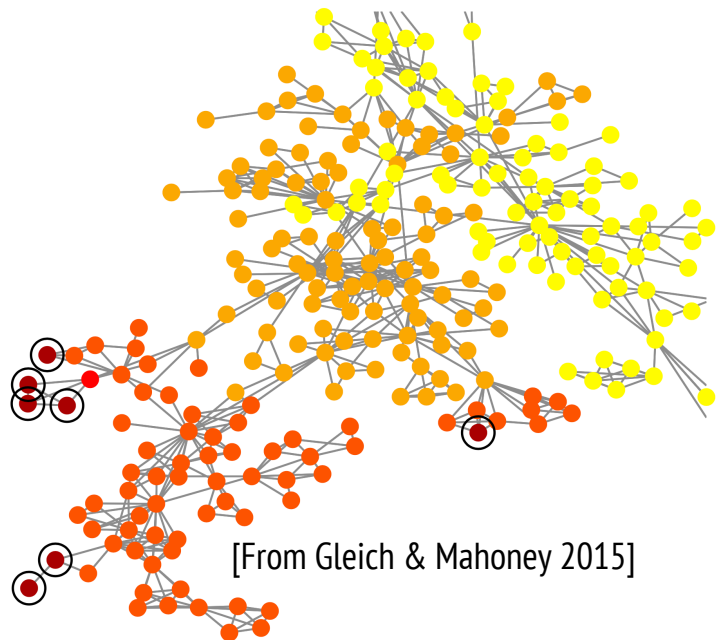
- Combine neighbor features via neural nets.
- Train with known target values.
- Produces vector \mathbf{h}_v for each node v .

Key questions.

1. When should each work well or poorly?
2. How can we combine them?
3. What is the relationship between them?

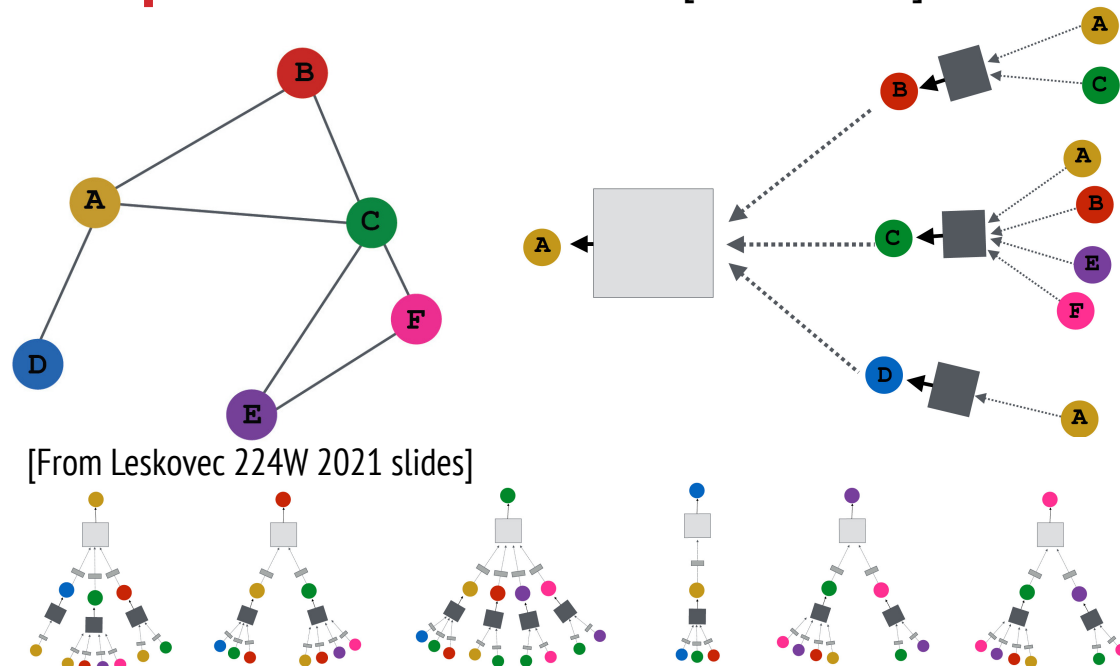
There are two broad classes of algorithms.

1. Label Propagation [early 2000s]



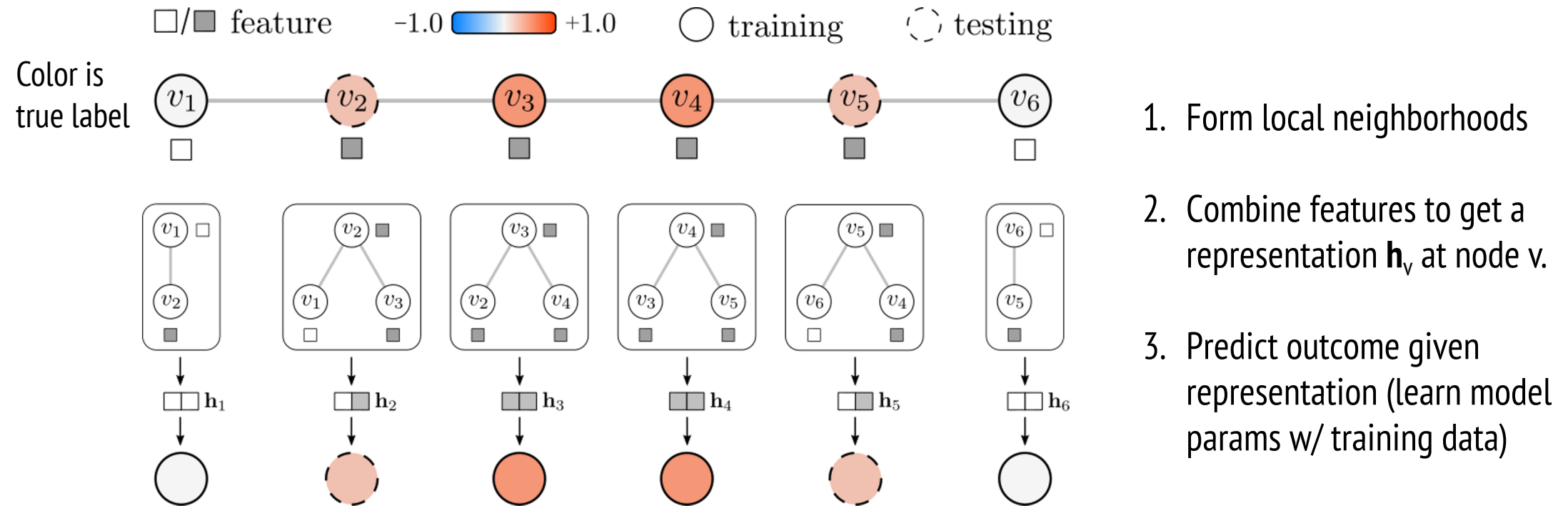
- Strong modeling assumption: connected nodes have similar labels.
- Works because of homophily [McPherson+ 01] a.k.a. assortativity [Newman 02]
- Why not use additional info/features?
- **FAST**
a few sparse matrix-vector products

2. Graph Neural Networks [late 2010s]



- Strong modeling assumption: labels only depend on neighbor features
- Works because these features are sometimes very informative.
- Why not assume labels are correlated?
- **SLOW**
many parameters, irregular computation

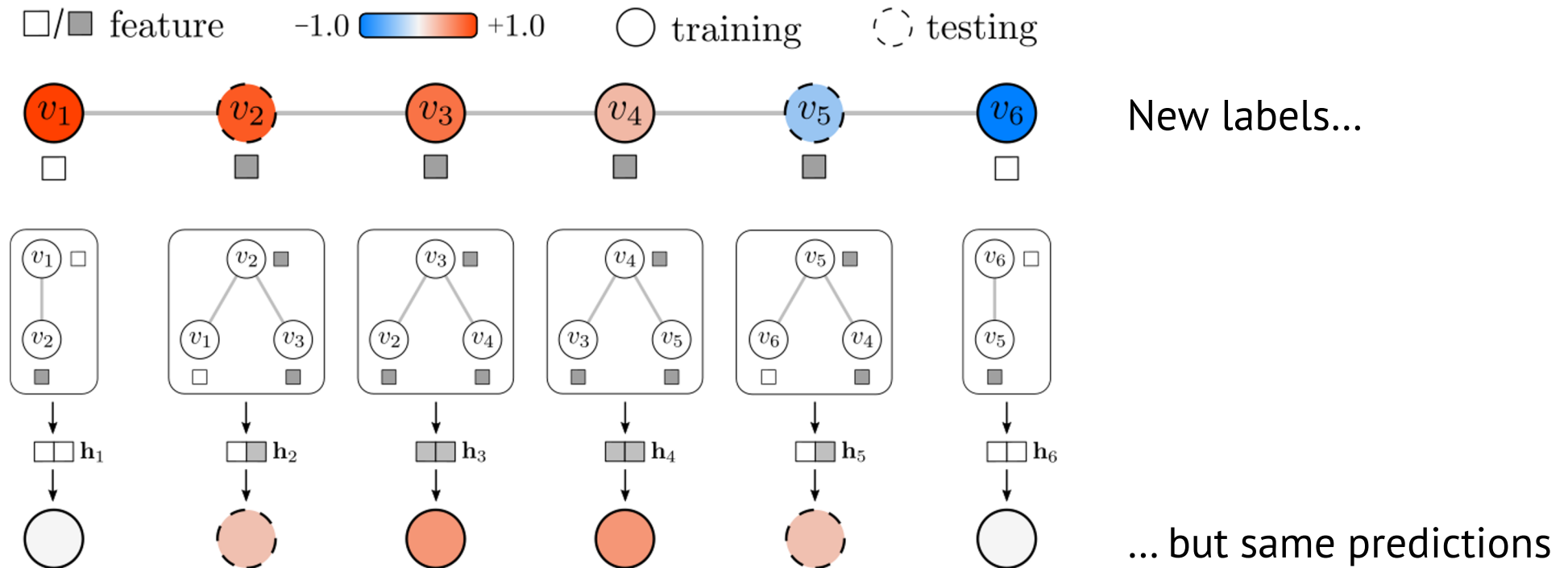
Graph neural networks make uncorrelated predictions.



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

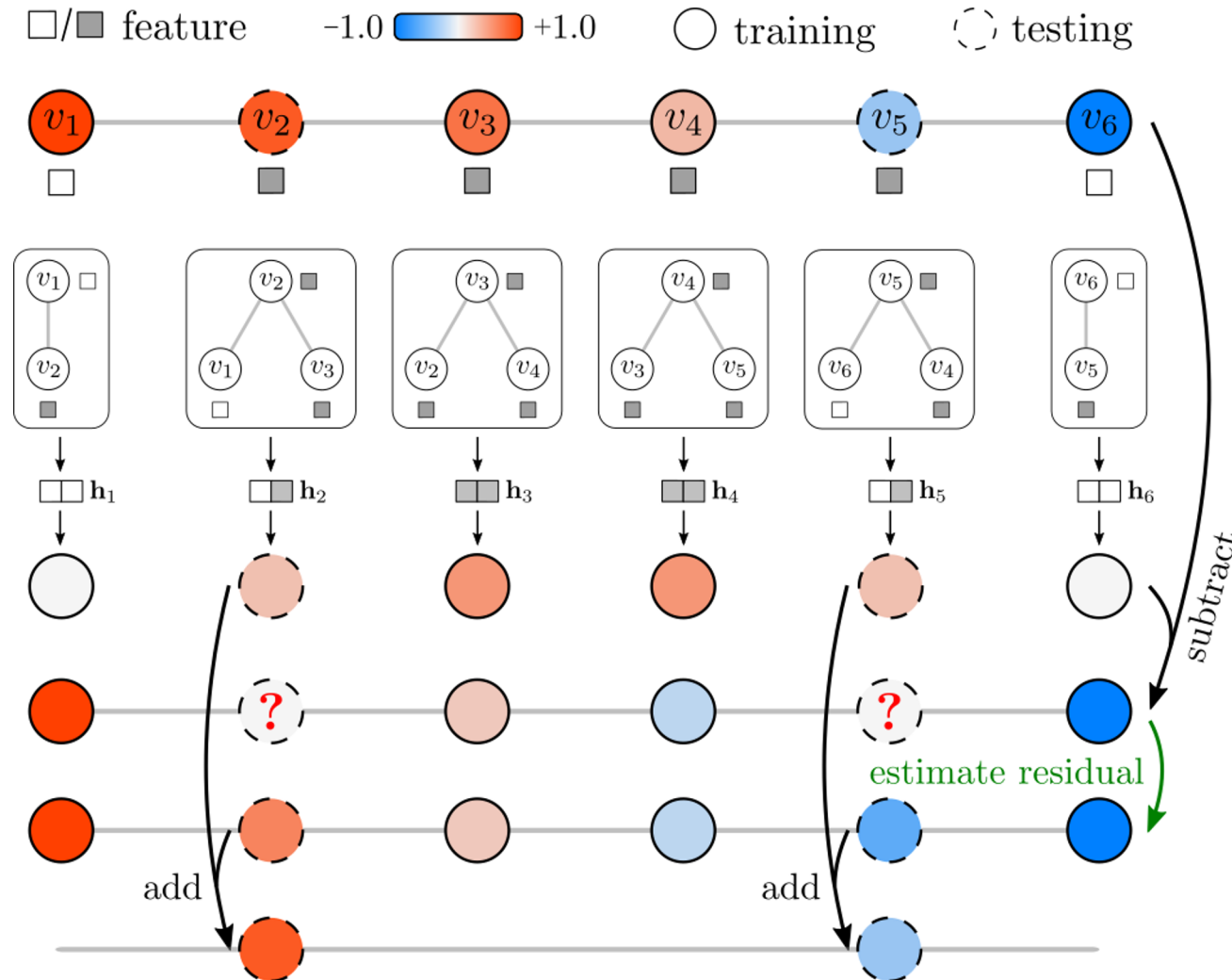
- Something strange? Given all \mathbf{h}_v , independent predictions.
- Use of the labels is very implicit.
- If node features are overwhelmingly predictive, this might be OK.

Uncorrelated GNN predictions can be catastrophic in simple cases when features are only mildly predictive.



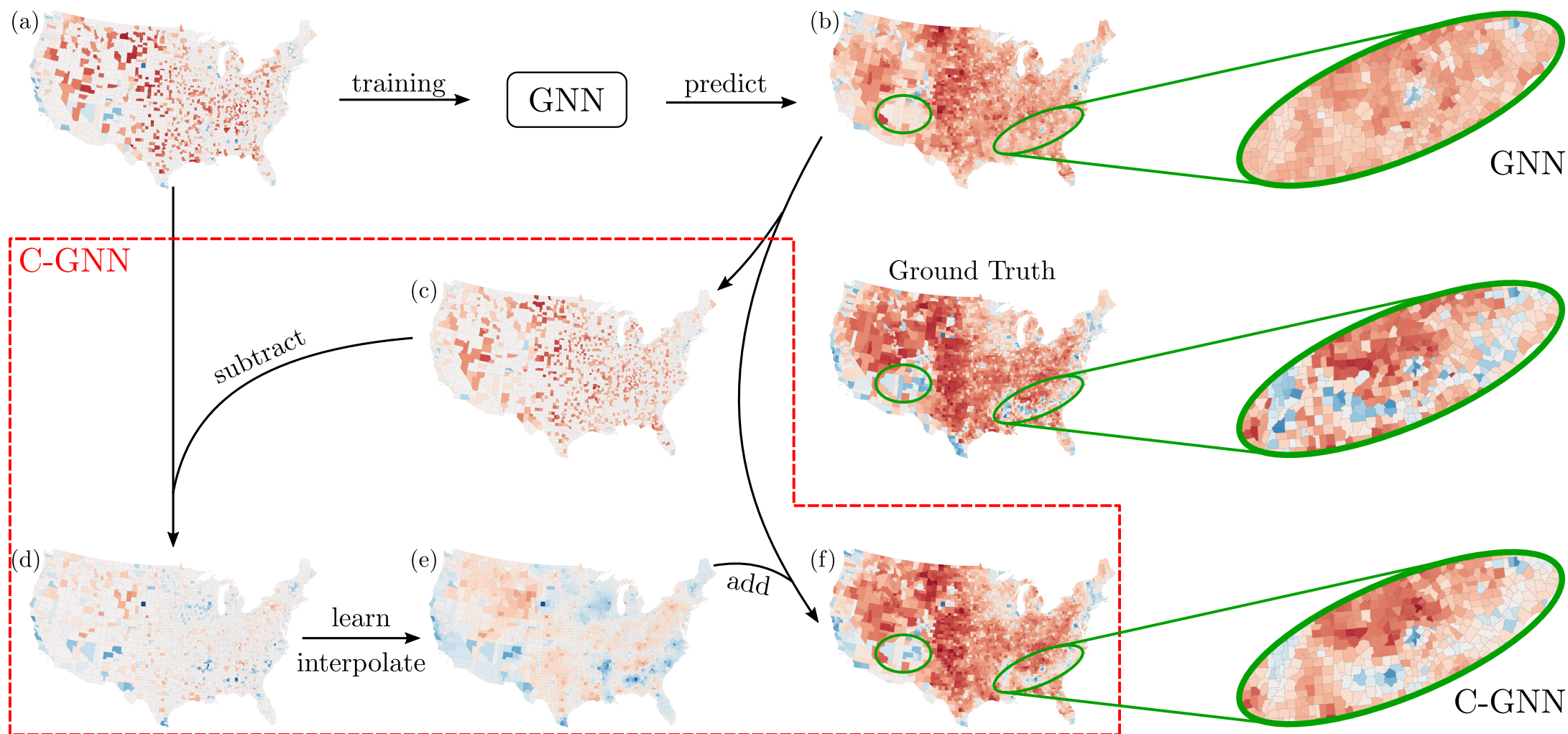
- All we have done is change the label distribution!
- **Big problem.** Features are no longer super predictive.
- LP (ignoring features) would work much better.

We can correlate feature-based predictions by propagating residual errors. [Jia-Benson 20, 21]



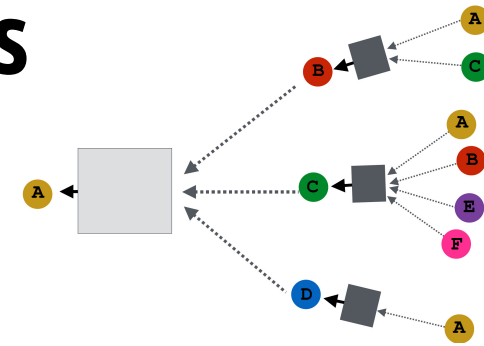
Works with any GNN.
Just layer on top.

Residual propagation works super well in practice.



Out-of-sample R^2 0.51 \rightarrow 0.69.

There is a simple statistical motivation for this problem with graph neural networks.



The objective is basically just ordinary least squares (OLS).

$$\min_{\theta} \sum_{u \in L} [y_u - g(\mathbf{x}_u, \{\mathbf{x}_v : v \in N_K(u)\}, \theta)]^2 = \min_{\theta} \sum_{u \in L} [y_u - \mathbf{h}_i(\theta)^T \beta(\theta)]^2$$

- If observations are $y_u = \beta^T \mathbf{h}_u + \varepsilon_u$ for i.i.d. $\varepsilon_u \sim N(0, \sigma^2)$, then the OLS solution is the MLE (also, Gauss-Markov theorem / BLUE).
- But we shouldn't expect i.i.d. error in graph data!
- We are positing that errors are positively correlated along edges.
- Essentially, this is a type of generalized least squares [Aitken 36].

Is there a common statistical
framework that unifies
LP and GNN ideas?

The formal problem we are solving.

Problem input.

- Graph $G = (V, E)$.
- $|V| \times p$ matrix \mathbf{X} of node features.
- Subset $L \subset V$ of labeled nodes.
- Length- $|L|$ vector \mathbf{y}_L of outcomes on L (real-valued or categorical).

Problem output.

- Length- $|U|$ vector \mathbf{y}_U of outcomes on $U = V \setminus L$.

Solution evaluation.

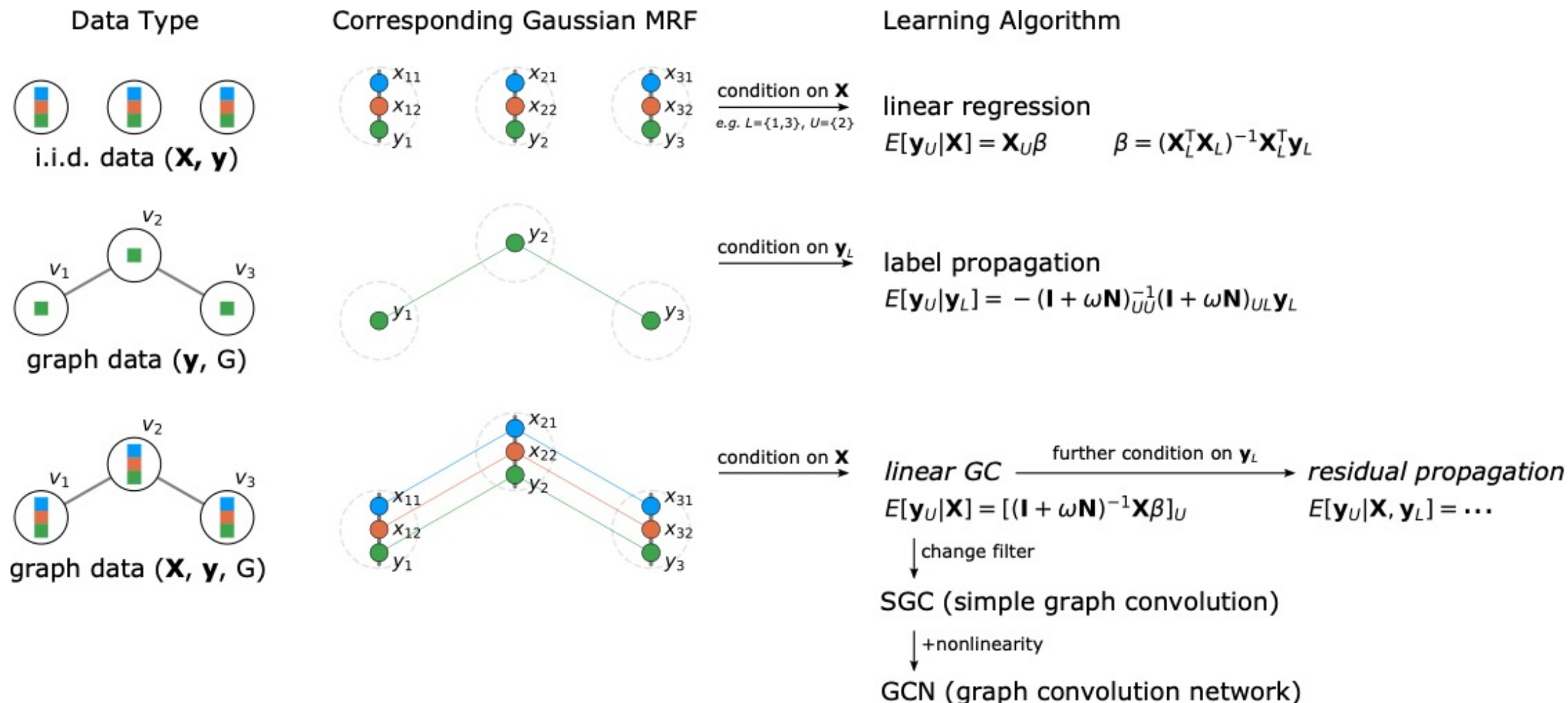
- Coefficient of determination $R^2 = 1 - \text{RSS} / \text{TSS}$.

Regression and classification.

- Most of the talk on theory and applications for *regression*.
Outcome variables like age, income, traffic flows, climate measurements.
- Later, extensions to *classification*.
Outcome variables like product or Web page categories / types.
(the theory is harder for classification, but we're working on it...)

We developed a random model for attributes on nodes, where statistical inference leads to GNN/LP algorithms.

[Jia-Benson 21]



Our model is based on smooth random attributes.

- Random real-valued attribute vectors $\mathbf{a}_u = [\mathbf{x}_u; y_u]$ on each node u .
- $\mathbf{A}_i = i$ th attribute over all nodes.
- $\mathbf{N} = \mathbf{I} - \mathbf{D}^{-1/2} \mathbf{W} \mathbf{D}^{-1/2}$ is the normalized Laplacian.
- Gaussian MRF random attribute model

$$\phi(\mathbf{A}|\mathbf{H}, \mathbf{h}) = \frac{1}{2} \sum_{u=1}^n \underbrace{\mathbf{a}_u^\top \mathbf{H} \mathbf{a}_u}_{\text{correlated attributes on a node}} + \frac{1}{2} \sum_{i=1}^{p+1} \underbrace{h_i \mathbf{A}_i^\top \mathbf{N} \mathbf{A}_i}_{\text{smoothness on attributes}}, \quad \mathbf{H} \in \mathbb{R}^{(p+1) \times (p+1)} \text{ spd}, \quad \mathbf{0} \leq \mathbf{h} \in \mathbb{R}^{(p+1)}$$

$$= \sum_{(u,v) \in E} (A_{ui}/\sqrt{d_u} - A_{vi}/\sqrt{d_v})^2$$

$$\rho(\mathbf{A} = \mathbf{A}' | \mathbf{H}, \mathbf{h}) = \frac{e^{-\phi(\mathbf{A}|\mathbf{H}, \mathbf{h})}}{\int d\mathbf{A}' e^{-\phi(\mathbf{A}'|\mathbf{H}, \mathbf{h})}} \quad \text{Smoother attributes are more likely (homophily / assortativity)}$$

$$\text{vec}(\mathbf{A}) \sim \mathcal{N}(\mathbf{0}, \mathbf{\Gamma}^{-1}), \quad \mathbf{\Gamma} = \mathbf{H} \otimes \mathbf{I}_n + \text{diag}(\mathbf{h}) \otimes \mathbf{N} \quad \text{Just a multivariate normal random variable in the end}$$

Now we can treat our graph learning problem as a statistical inference problem.

Problem input.

- Graph $G = (V, E)$.
- $|V| \times p$ matrix \mathbf{X} of node features.
- Subset $L \subset V$ of labeled nodes.
- Length- $|L|$ vector \mathbf{y}_L of outcomes on L (real-valued or categorical).

Problem output.

- Length- $|U|$ vector \mathbf{y}_U of outcomes on $U = V \setminus L$.

Problem solution.

- $\mathbf{y}_U = E[\mathbf{y}_U \mid \text{input data}]$, under our model
- Different conditioning \rightarrow different algorithms.

Case 1. Linear regression when there are no edges.

(special case of standard theory of linear models)

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

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

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

rather than infer directly, estimate with OLS

Case 2. Label propagation when conditioning on observed labels, assuming no features.

One attribute, so these are just positive scalars

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

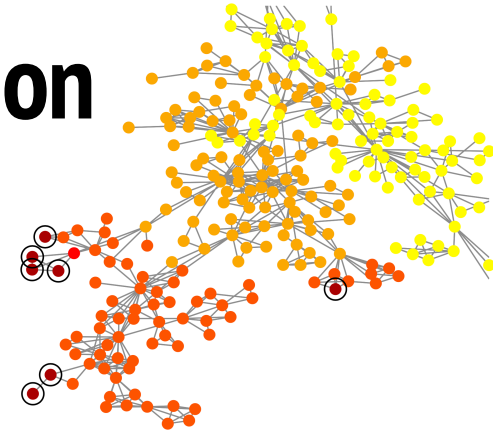
control noise, $1/H$ is variance if no edges

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

control smoothness

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

Case 2. Label propagation when conditioning on observed labels, assuming no features.



$E[\mathbf{y}_U | \mathbf{y}_L = \mathbf{y}_L] = -(\mathbf{I}_n + \omega \mathbf{N})_{UU}^{-1} (\mathbf{I}_n + \omega \mathbf{N})_{UL} \mathbf{y}_L$ is the limit of label propagation.

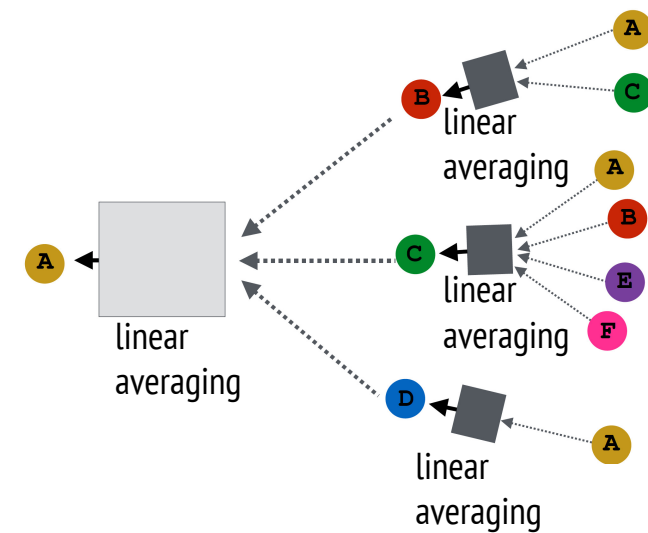
$$\forall u \in U, y_u^{(t+1)} \leftarrow (1 - \alpha) \cdot y_u^{(0)} + \alpha \cdot d_u^{-\frac{1}{2}} \sum_{v \in N_1(u)} d_v^{-\frac{1}{2}} y_v^{(t)}; \quad \forall u \in L, y_u^{(t+1)} \leftarrow y_u^{(t)}$$

$$\forall u \in U, y_u^{(0)} = 0; \quad \forall u \in L, y_u^{(0)} = y_u$$

$$\alpha = \omega / (1 + \omega) = h / H / (1 + h / H) \in (0, 1)$$

- $h \nearrow \rightarrow \text{smoothness} \nearrow \rightarrow \omega \nearrow \rightarrow \alpha \nearrow \rightarrow$ more weight on neighbors
- $h \searrow \rightarrow \text{smoothness} \searrow \rightarrow \omega \searrow \rightarrow \alpha \searrow \rightarrow$ less weight on neighbors
- $H \nearrow \rightarrow \text{noise} \searrow \rightarrow \omega \searrow \rightarrow \alpha \searrow \rightarrow$ less weight on neighbors
- $H \searrow \rightarrow \text{noise} \nearrow \rightarrow \omega \nearrow \rightarrow \alpha \nearrow \rightarrow$ more weight on neighbors
- Rather than infer directly, estimate α with cross validation

Case 3. Linear graph convolutions when conditioning on features.



$$\mathbf{y}|\mathbf{X} = \mathbf{X} \sim \mathcal{N}(\bar{\mathbf{y}}, \Gamma_{PP}^{-1}), \quad \Gamma_{PP} = \underbrace{H_{p+1,p+1}}_{\text{control noise (inverse variance)}} \mathbf{I}_n + \underbrace{h_{p+1}}_{\text{control smoothness}} \mathbf{N}$$

$$\begin{aligned} \bar{\mathbf{y}} &= E[\mathbf{y}|\mathbf{X} = \mathbf{X}] = (H_{p+1,p+1} \mathbf{I}_n + h_{p+1} \mathbf{N})^{-1} (-\mathbf{H}_{1:p,p+1}^T \otimes \mathbf{I}_n) \text{vec}(\mathbf{X}) \\ &= \underbrace{(\mathbf{I}_n + \omega \mathbf{N})^{-1} \mathbf{X}}_{\text{LP on features! (just no known, fixed values)}} \boldsymbol{\beta} \end{aligned}$$

LP on *features!* (just no known, fixed values)

$$\begin{aligned} \forall v \in V, \quad x_v^{(t+1)} &\leftarrow (1 - \alpha) \cdot x_v^{(0)} + \alpha \cdot d_v^{-\frac{1}{2}} \sum_{w \in N_1(v)} d_w^{-\frac{1}{2}} x_w^{(t)} \\ \forall v \in V, \quad x_v^{(0)} &= x_v \end{aligned}$$

Linear graph convolution (LGC).

1. Run LP on each feature \rightarrow smoothed features.
2. OLS on these preprocessed, smoothed features.

Case 3. Linear graph convolutions when conditioning on features.

Linear Graph Convolution (LGC) $(1 - \alpha) (I + \alpha S + \alpha^2 S^2 + \dots) X \beta$ $S = D^{-1/2} W D^{-1/2}$
[Jia-Benson 21]

Simplified Graph Convolution (SGC) $\tilde{S}^K X \beta$ $\tilde{S} = (D + I)^{-1/2} (W + I) (D + I)^{-1/2}$
[Wu+ 19]

Graph Convolution Network (GCN) $\sigma(\tilde{S} \dots \sigma(\tilde{S} X \Theta^{(1)}) \dots \Theta^{(K)}) \beta$
[Kipf-Welling 17]

- α is continuous, while K is discrete.
- SGC as $K \rightarrow \infty$ is nonsensical.
- Does nonlinearity help?
- Does extra parameterization of each “propagation step” in GCN help?
- *No conditioning on label distribution!*

We should condition on
features *and* labels in our model.

And GNN predictions should not be
conditionally independent!

Case 4. Residual propagation when conditioning on both features and observed labels.

$$\bar{\mathbf{y}} = (\mathbf{I}_n + \omega \mathbf{N})^{-1} \mathbf{X} \beta$$

LP on *residual errors*!

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

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

1. Run LP on each feature \rightarrow smoothed features.
2. OLS on these smoothed features \rightarrow initial predictions.
3. Run LP on residual errors \rightarrow smoothed errors.
4. Add smoothed errors to initial predictions.

Can substitute in any initial prediction.

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

We can examine regression coefficients with LGC.

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

Outcome = republican vote share – 0.5

Zero mean / unit variance feature normalization

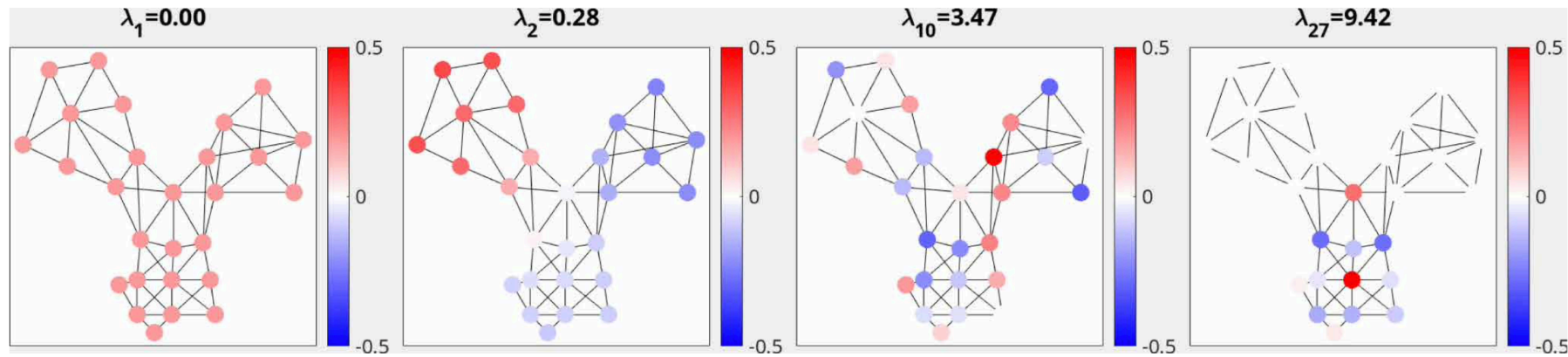
- Higher income and lower education levels → right-leaning
- Income and education level stronger indicators in 2016.
- Positive sh500m coefficient from rural, right-leading counties?

We can also evaluate on our generative model.

	h_0	LP (α)	LR	LGC (α)	SGC (K)	GCN (K)	LGC/RP (α)	SGC/RP (K, α)	GCN/RP (K, α)
Low homophily.	1	0.19 (0.79)	0.68	0.70 (0.28)	0.37 (1.8)	0.34 (1.7)	0.73 (0.29)	0.40 (1.8, 0.21)	0.37 (1.7, 0.21)
	10	0.43 (0.95)	0.48	0.58 (0.57)	0.45 (2.1)	0.45 (2.0)	0.68 (0.56)	0.56 (2.1, 0.46)	0.54 (2.0, 0.43)
High homophily.	100	0.59 (0.99)	0.24	0.42 (0.85)	0.38 (2.3)	0.45 (2.5)	0.64 (0.85)	0.63 (2.3, 0.81)	0.62 (2.5, 0.79)

- GCN more expressive but prone to overfitting.
- More homophily \rightarrow larger K, α
- Adding residual prop never hurts!
- GCN better with more homophily?
“memorizing” neighborhood features (zero training error)
+ smoothness in data \rightarrow better out-of-sample prediction

Our model helps us understand smoothing.



Graph Signal Processing:
Overview, Challenges and
Applications, Ortega et al.,
Proc. IEEE, 2018.

$$\mathbf{N} = \mathbf{V} \mathbf{\Lambda} \mathbf{V}^T, \text{ feature } \mathbf{f} = \sum_{i=1}^n c_i \mathbf{v}_i$$

$$\text{LGC } \mathbf{f} \rightarrow \sum_{i=1}^n \frac{1}{(1 + \omega \lambda_i)} c_i \mathbf{v}_i$$

$$\text{SGC } \mathbf{f} \rightarrow \sum_{i=1}^n (1 - d/(d+1)\lambda_i)^K c_i \mathbf{v}_i$$

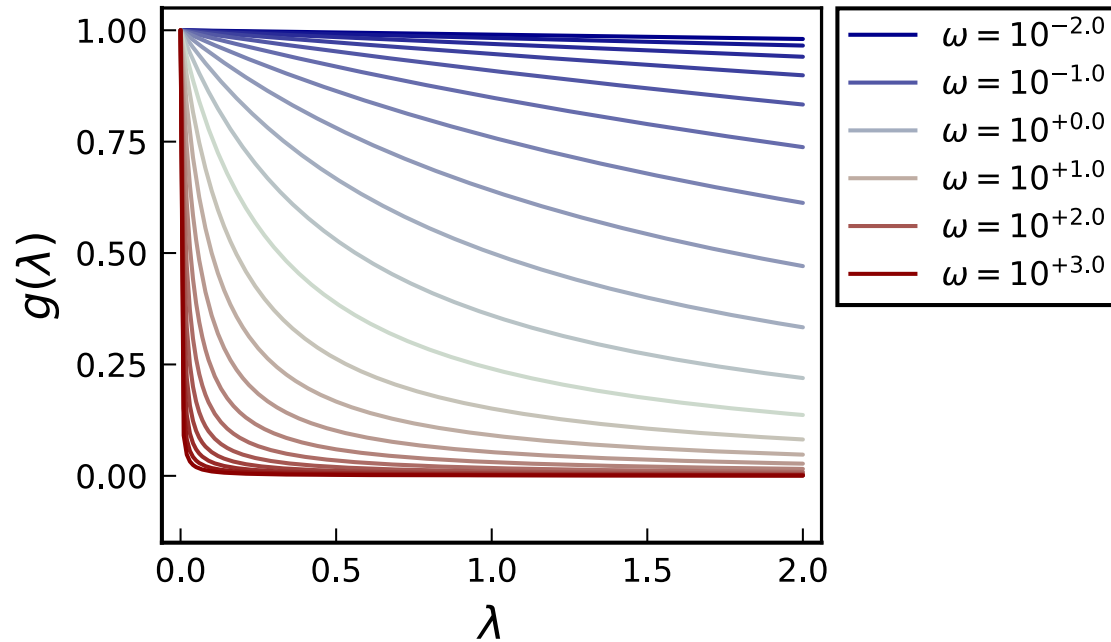
Low-pass on $[0, \infty)$,
continuous parameterization.

Low-pass on $[0, (d + 1)/d]$,
discrete parameterization.

Encouraging
smoothness.

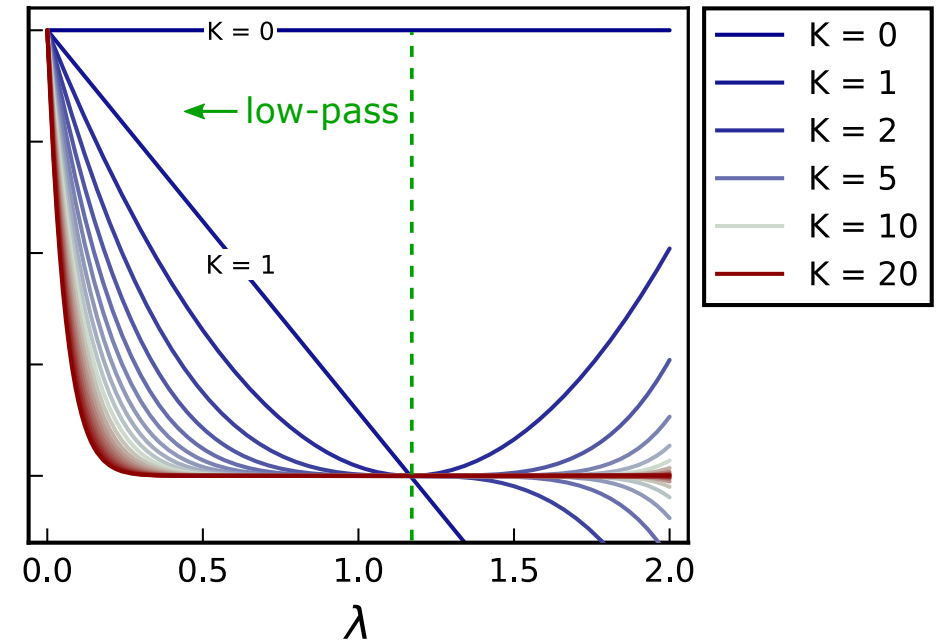
Our model helps us understand smoothing.

LGC



$$\mathbf{f} = \sum_{i=1}^n c_i \mathbf{V}_i \rightarrow \sum_{i=1}^n \frac{1}{(1 + \omega \lambda_i)} c_i \mathbf{V}_i$$

SGC

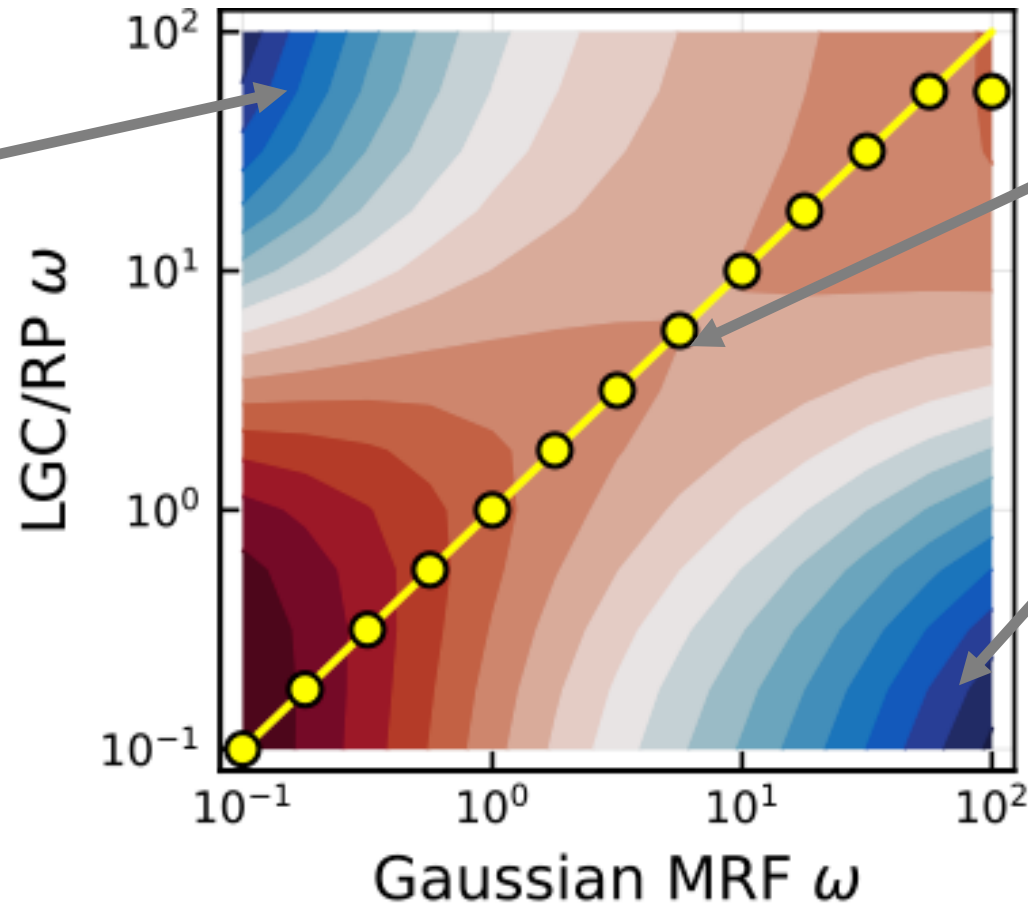


$$\mathbf{f} = \sum_{i=1}^n c_i \mathbf{V}_i \rightarrow \sum_{i=1}^n (1 - d/(d+1) \lambda_i)^K c_i \mathbf{V}_i$$

Our model helps us understand smoothing.

“Oversmoothing.”

Empirically discussed
problem with GNNs.
[Li+ 18; Oono-Suzuki 20;
Zhou-Akoglu 20]



CV can identify the
model parameter.

Undersmoothing?

Possible but not
discussed in the
literature.

Our model provides a nice setup for inductive learning.

Problem input.

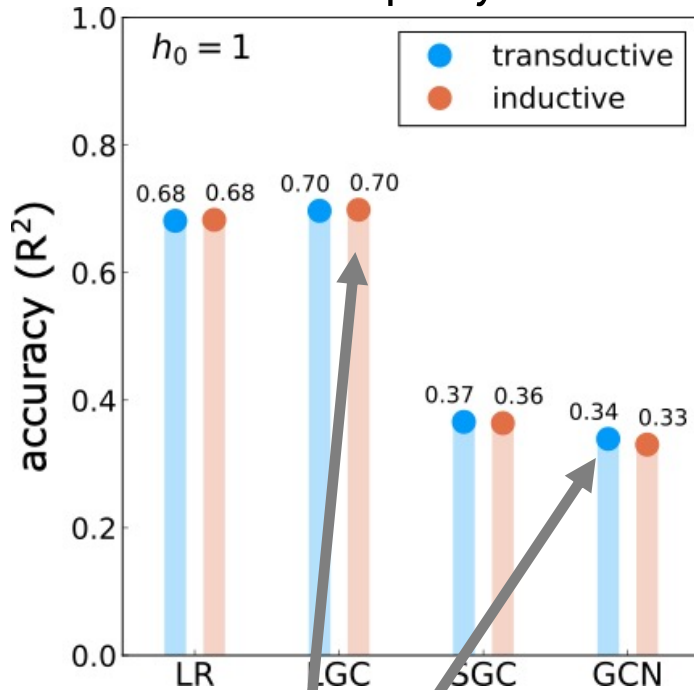
- Graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$.
- $|V_1| \times p$ matrix \mathbf{X}_1 and $|V_2| \times p$ matrix \mathbf{X}_2 of node features (same features)
- Subset $L_1 \subset V$ of labeled nodes.
- Length- $|L_1|$ vector \mathbf{y}_{L_1} of outcomes on L_1 .

Problem output.

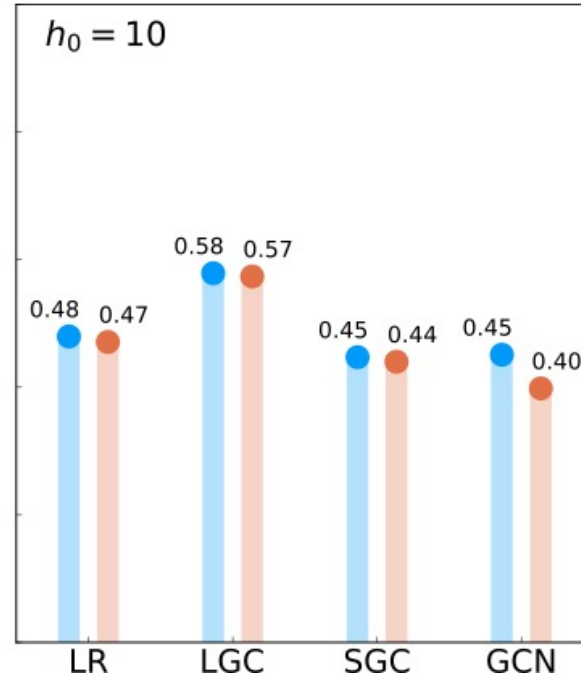
- Length- $|V_2|$ vector \mathbf{y} of outcomes on nodes V_2 .

Our model provides a nice setup for inductive learning.

Predictive features,
low homophily.

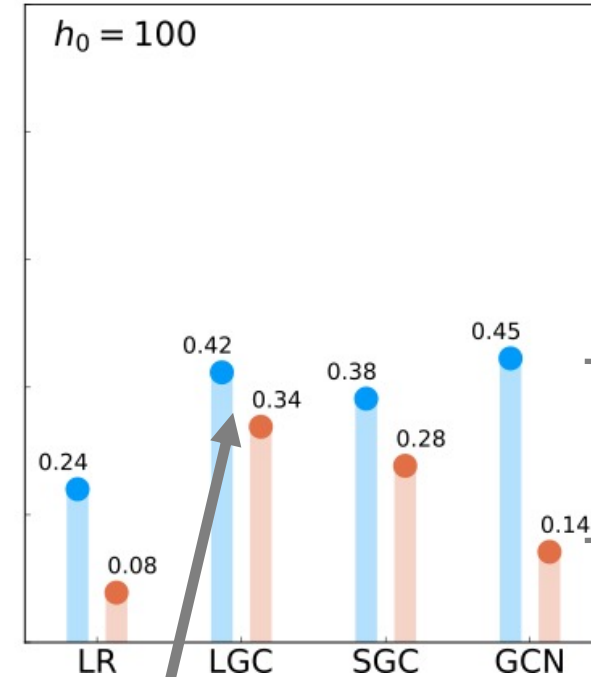


No performance
degradation.



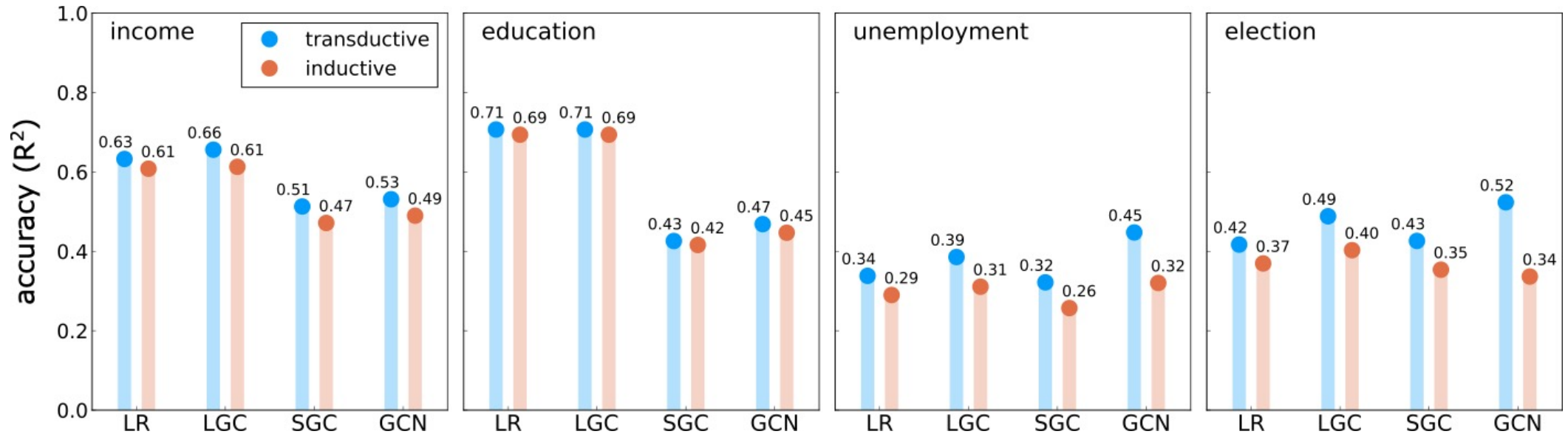
A bit of degradation.

High homophily.



Bad
overfitting!

Our model provides a nice setup for inductive learning.



- Graph G_1 from 2012 election data.
- Graph G_2 from 2016 election data.

Label propagation is a powerful tool.

1. LP can be effectively applied to features (smoothing / de-noising).
2. LP is fast and scalable... just need averaging of neighbors.
3. LP and GNN ideas can come from a common model.
4. Linear models are often superior to nonlinear ones (GNNs) in practice.
5. Residual propagation is super effective (smooth errors), and can also be implemented with just the LP primitive.
6. Residual propagation can be added to any base predictor, and we never really see it hurt performance.

An aside:
We ❤️ julia

```
41 function interpolate(L, rL;  $\Gamma$ )
42     """
43     Args:
44         L: mini_batch indices for estimating noise
45         rL: noise over the mini_batch L
46          $\Gamma$ : label propagation matrix
47
48     Returns:
49         r: noise over all vertices
50     """
51     n = size( $\Gamma$ ,1);
52     U = setdiff(1:n, L);
53     rU = vcat([c $\Gamma$ ( $\Gamma$ [U,U], - $\Gamma$ [U,L]*rL[i,:])' for i in 1:size(rL,1)]...);
54
55     r = zeros(size(rL,1), n);
56     r[:,L] = rL;
57     r[:,U] = rU;
58
59     return r;
60 end
61
62 function estimate_residual(U, L; LBL, pL,  $\Gamma$ )
63     """
64     Args:
65         U: vertices to predict
66         L: vertices with ground truth labels
67         LBL: ground truth labels on L
68         pL: predicted labels
69          $\Gamma$ : label propagation matrix
70
71     Returns:
72         lU: predictive label = base predictor output + estimated noise
73     """
74
75     # for regression task, residual is defined as ``true-label minus predicted-label``
76     rL = LBL - pL;
77     rU = interpolate(L, rL;  $\Gamma$ = $\Gamma$ )[:,U];
78
79     return rU;
80 end
```

<https://github.com/000Justin000/GaussianMRF/blob/main/predict.jl>

Where do we go from here?

1. What if outcomes are not positively correlated?
(non-homophily, disassortativity)
2. What if outcomes are categorical?
(classification problems)

We can also learn the correlation directly. [Jia-Benson 20]

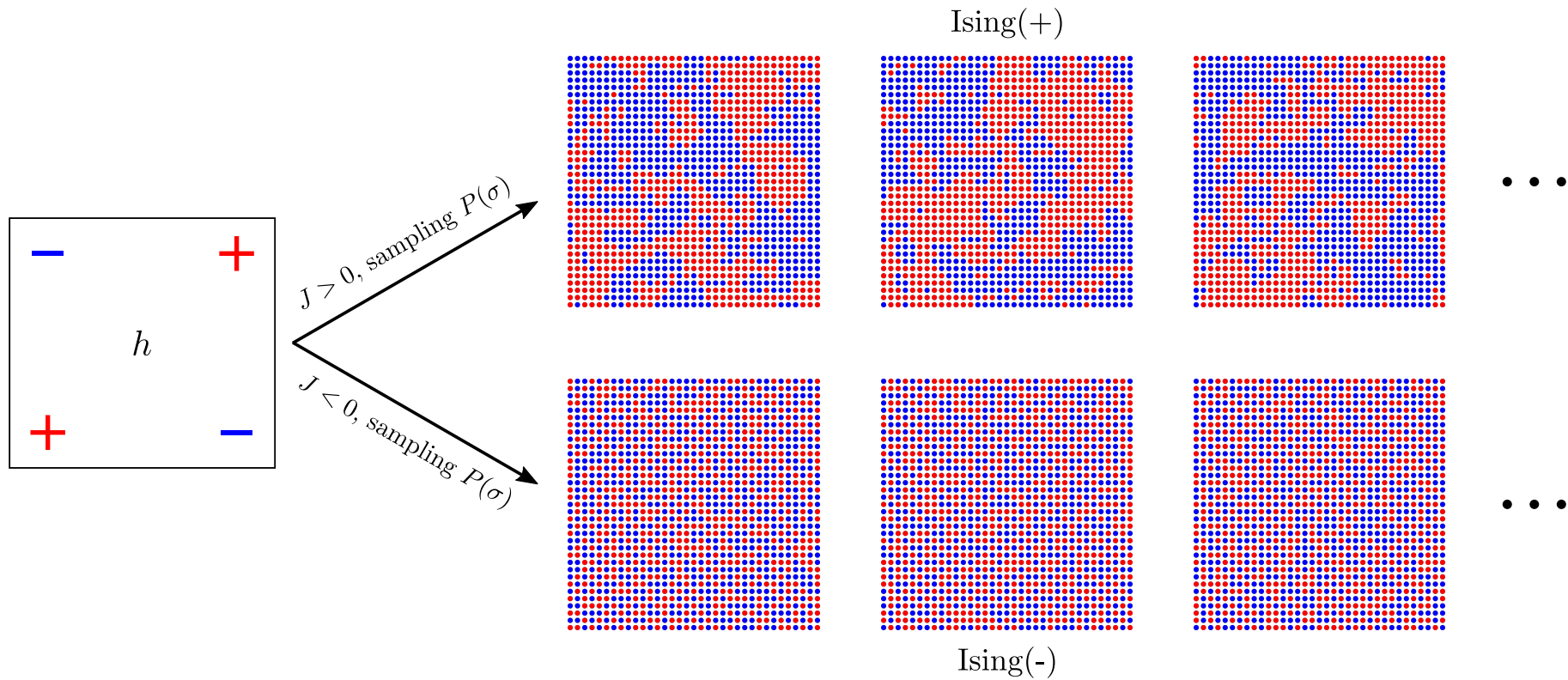
$$\text{residual} \sim N(\mathbf{0}, \mathbf{\Gamma}^{-1}), \quad \mathbf{\Gamma} = \beta(\mathbf{I} - \alpha \mathbf{S}), \quad \mathbf{S} = \mathbf{D}^{-1/2} \mathbf{W} \mathbf{D}^{-1/2}$$

- $\beta > 0$ is correlation strength
- $\alpha = 0 \rightarrow$ uncorrelated outcomes (no residual prop)
- $\alpha > 0 \rightarrow$ positively correlated outcomes
- $\alpha < 0 \rightarrow$ negatively

Jointly maximize likelihood of base model + correlated error.

Requires lots of numerical tricks to be scalable.

We can learn negative correlations.



features are grid coordinates

Dataset	GCN	GCN + standard RP	learned corr. RP
Ising(+)	0.61 ± 0.04	0.72 ± 0.03	0.72 ± 0.03
Ising(-)	0.47 ± 0.02	0.34 ± 0.02	0.70 ± 0.03

A simple binary classification extension

- Positive class \rightarrow value 1.0
- Negative class \rightarrow value 0.0
- Threshold for classification

dataset	LP	LR	LGC	SGC	GCN	LGC/RP	SGC/RP	GCN/RP
Elliptic	0.70	0.62	0.61	0.59	0.79	0.71	0.73	0.83

More classification, and how far we can get without GNNs.



Joint work with Qian Huang, Horace He, Abhay Singh (Cornell undergrads), and Ser-Nam Lim (FB)



[Get Started](#) [Updates](#) [Datasets ▾](#) [Leaderboards ▾](#) [Paper](#) [Team](#) [Github](#)

Leaderboard for [ogbn-products](#)

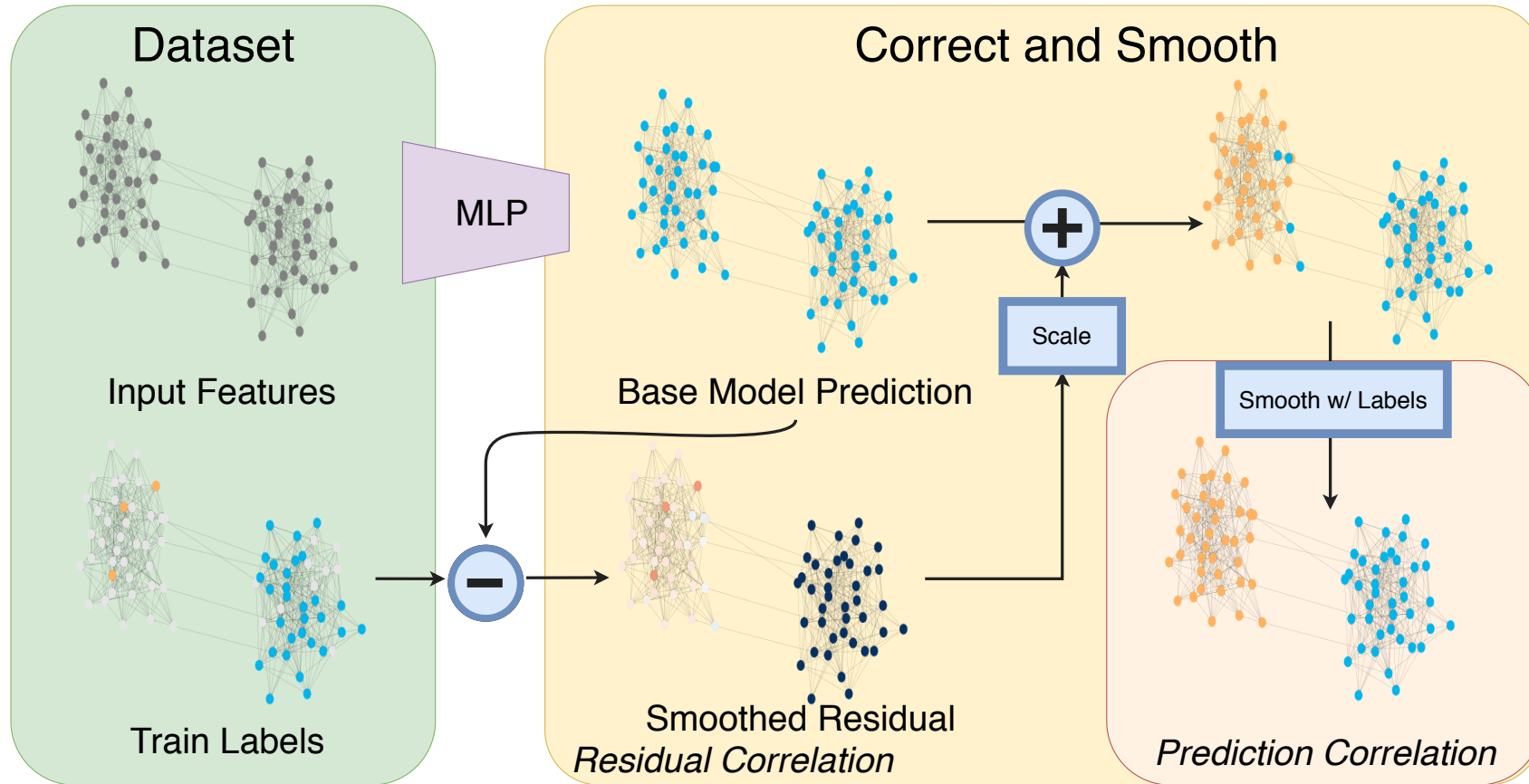
The classification accuracy on the test and validation sets. The higher, the better.

Package: $\geq 1.1.1$

Rank	Method	Test Accuracy	Validation Accuracy	Contact	References	#Params	Hardware	Date
1	MLP + C&S	0.8418 \pm 0.0007	0.9147 \pm 0.0009	Horace He (Cornell)	Paper , Code	96,247	GeForce RTX 2080 (11GB GPU)	Oct 27, 2020
2	Linear + C&S	0.8301 \pm 0.0001	0.9134 \pm 0.0001	Horace He (Cornell)	Paper , Code	10,763	GeForce RTX 2080 (11GB GPU)	Oct 27, 2020
3	UniMP	0.8256 \pm 0.0031	0.9308 \pm 0.0017	Yunsheng Shi (PGL team)	Paper , Code	1,475,605	Tesla V100 (32GB)	Sep 8, 2020
4	Plain Linear + C&S	0.8254 \pm 0.0003	0.9103 \pm 0.0001	Horace He (Cornell)	Paper , Code	4,747	GeForce RTX 2080 (11GB GPU)	Oct 27, 2020
5	DeeperGCN+FLAG	0.8193 \pm 0.0031	0.9221 \pm 0.0037	Kezhi Kong	Paper , Code	253,743	NVIDIA Tesla V100 (32GB GPU)	Oct 20, 2020

We can get great classification accuracy without GNNs.

(using ideas of residual prop / smoothness, but with some fiddling)



The formal problem we are solving.

Problem input.

- Graph $G = (V, E)$.
- $|V| \times p$ matrix \mathbf{X} of node features.
- Subset $L \subset V$ of labeled nodes.
- Length- $|L|$ vector \mathbf{y}_L of categorical outcomes (classes) on L .

Problem output.

- Length- $|U|$ vector \mathbf{y}_U of categorical outcomes (classes) on $U = V \setminus L$.

Solution evaluation.

- Accuracy = fraction of entries in \mathbf{y}_U that are correct labels.

We get a lot of mileage out of smoothness.

1. Base predictions.

Base predictor \rightarrow a vector of class probabilities \mathbf{p}_u at each node u .
Use logistic reg. or multilayer perceptron (MLP) via softmax.
No use of graph yet but could use.

2. Residual prop adaptation (correction step).

Form error vector $\mathbf{e}_u = \text{one_hot}(\mathbf{y}_u) - \mathbf{p}_u$ at labeled nodes.
Run residual prop on each component \rightarrow smoothing residual vector \mathbf{r}_v .
Scaling not quite right, correct with $\mathbf{z}_v = \mathbf{p}_v + s * \mathbf{r}_v$ (tune scalar s).

3. Reset on labeled nodes.

$\mathbf{z}_u = \text{one_hot}(\mathbf{y}_u)$ if u labeled; \mathbf{z}_v same if v unlabeled (not a probability, though).

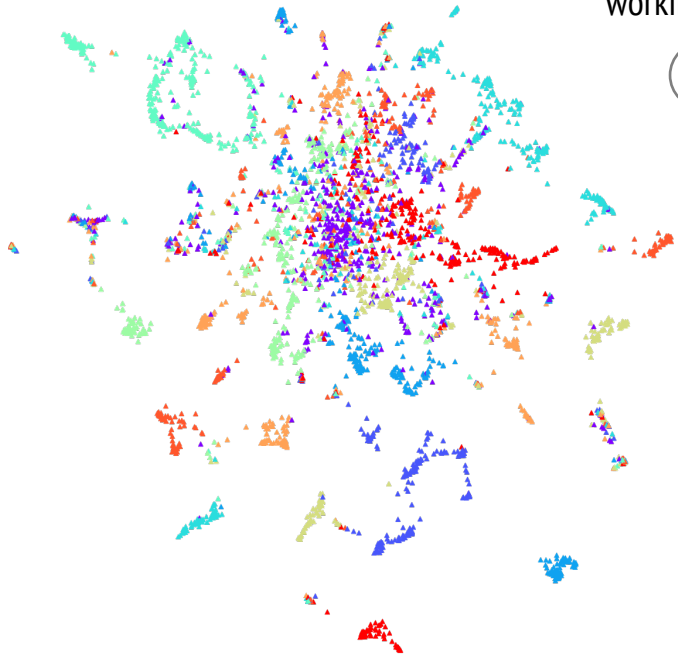
4. Smooth corrected vectors (smooth step).

Run LP on each coordinate of \mathbf{z} to get smoothed vectors \mathbf{y}_v .

5. Final predictions.

For unlabeled node v , predict maximum entry in \mathbf{y}_v .

Ground truth clusters

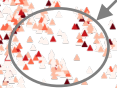


Log. reg. + Correct & Smooth

Linear model
working well



Extra smoothing
helping on less
obvious cases.

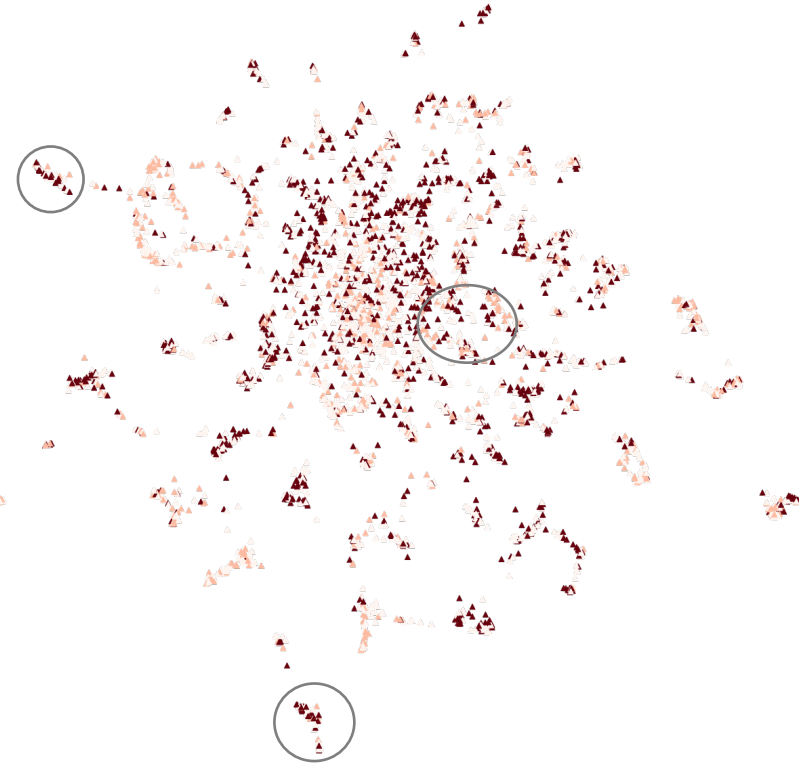


Training
Base
Base + C
Base + C&S
Mistake



Residual correlation
helping on periphery

GCN++



We get better accuracy with fewer parameters while being much faster to train.

Datasets	Classes	Nodes	Edges	Parameter Δ	Accuracy Δ	Time
Products	47	2,449,029	61,859,140	-93.47%	+1.53	170.6 s
Arxiv	40	169,343	1,166,243	-84.9%	+ 0.97	9.89 s
Cora	7	2,708	5,429	-98.37%	+ 1.09	0.5 s
Citeseer	6	3,327	4,732	-89.68%	- 0.69	0.48 s
Pubmed	3	19,717	44,338	-96.00%	- 0.30	0.85 s
Email	42	1,005	25,571	- 97.89%	+ 4.26	42.83 s
Rice31	10	4,087	184,828	- 99.02%	+ 1.39	39.33 s
US County	2	3,234	12,717	- 74.56%	+ 1.77	39.05 s
wikiCS	10	11,701	216,123	- 84.88%	+ 2.03	7.09 s



A few extra things that sometimes help...

1. Augment node features (spectral embedding, motif counts).
2. Pre-trained big GNNs for base classifiers (often hurts, though).
3. Other types of scaling of residual prop output.



Label propagation should be a standard tool in graph-based learning for estimating node attributes.

1. LP is a classical smoothing / correlation technique.
LP helps denoise features, correlate errors, smooth final predictions
2. While traditionally seen as separate ideas, LP and basic GNN ideas can be derived from a common model and combined effectively.
3. #IJALM
$$\mathbf{y}_U^{\text{LGC/RP}} = [(\mathbf{I}_n + \omega \mathbf{N})^{-1} \mathbf{X} \boldsymbol{\beta}]_U - (\mathbf{I} + \omega \mathbf{N})_{UU}^{-1} (\mathbf{I} + \omega \mathbf{N})_{UL} (\mathbf{y}_L - [(\mathbf{I}_n + \omega \mathbf{N})^{-1} \mathbf{X} \boldsymbol{\beta}]_L)$$
4. Lots of GNN variants... other generative models?
5. Theory or more principled approaches for classification?
6. Similar ideas for other graph problems? Link prediction...


Label Propagation and Graph Neural Networks

THANKS! Austin R. Benson
<http://cs.cornell.edu/~arb>
 @austinbenson
 arb@cs.cornell.edu

A Unifying Generative Model for Graph Learning Algorithms: Label Propagation, Graph Convolutions, and Combinations.
Junteng Jia and Austin R. Benson. arXiv:2101.07730, 2021.

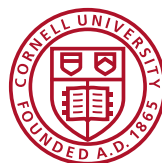
  <https://github.com/000Justin000/GaussianMRF>

Residual Correlation in Graph Neural Network Regression.
Junteng Jia and Austin R. Benson. Proc. of KDD, 2020.

  <https://github.com/000Justin000/gnn-residual-correlation>

Combining Label Propagation and Simple Models Out-performs Graph Neural Networks.
Qian Huang, Horace He, Abhay Singh, Ser-Nam Lim, and Austin R. Benson. Proc. of ICLR, 2020.

  <https://github.com/CUAI/CorrectAndSmooth>



Cornell University