Label Propagation and Graph Neural Networks

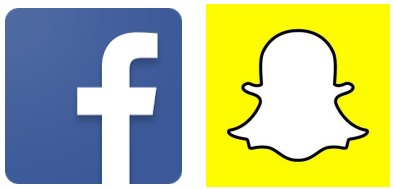
Austin R. Benson · Cornell University

Network Data Science 2 (PHYS 7332), Northeastern · March 12, 2021



Joint work with
Junteng Jia, Qian Huang, Horace He, & Abhay Sing (Cornell)
Ser-Nam Lim (Facebook)

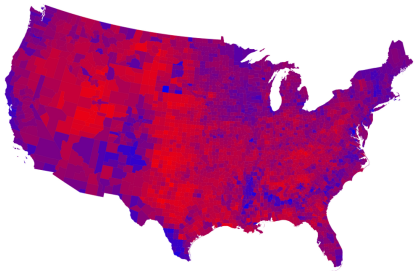
Graph data modeling complex systems are everywhere.



Society
nodes are people
edges are friendships



Finance
nodes are accounts
edges are transactions



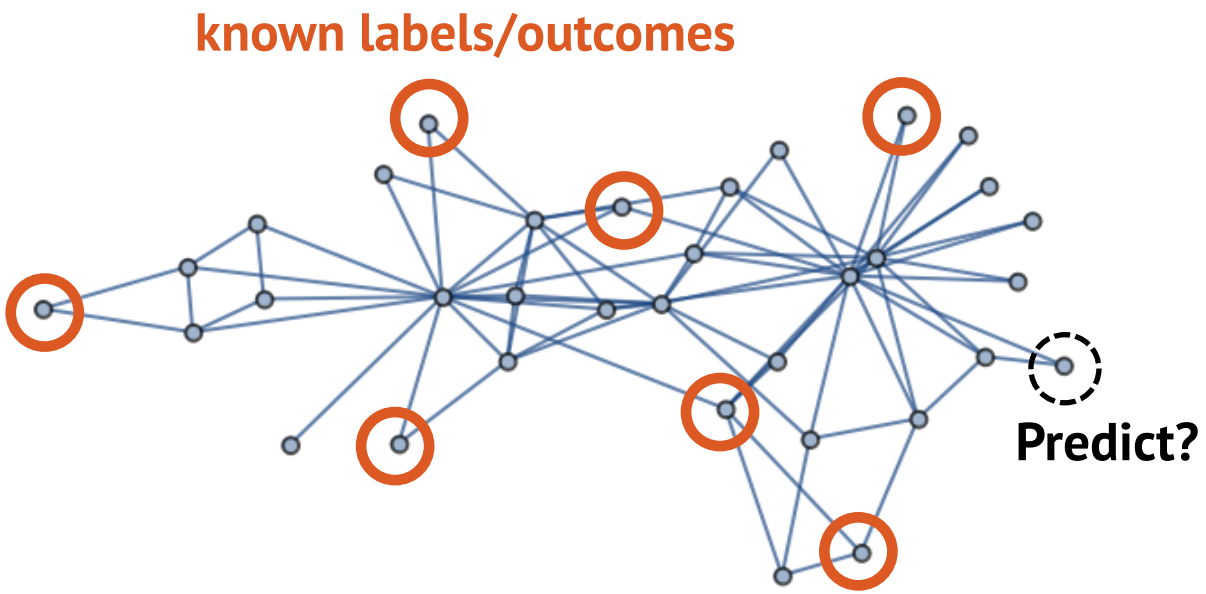
[Mark Newman 2012 map]

Elections
nodes are regions
edges are social / geo



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.

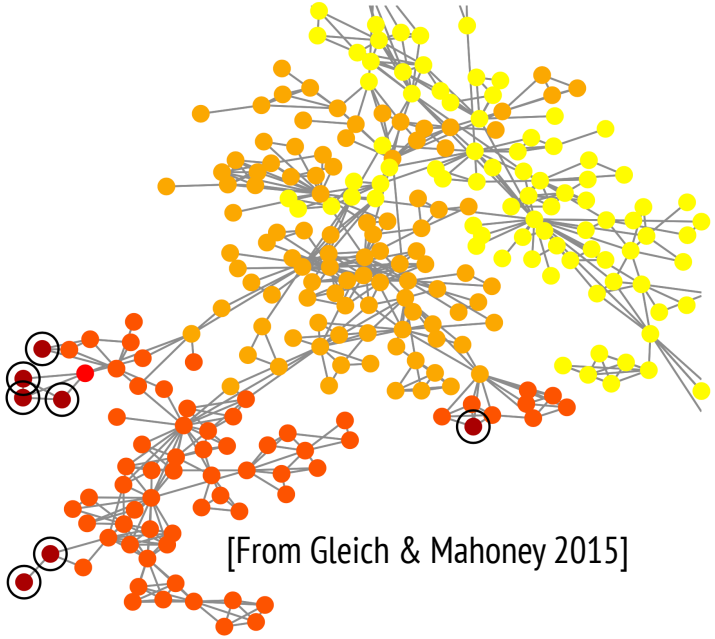
- Graph $G = (V, E)$.
- $|V|$ x p matrix $\boldsymbol{X}$ of node features.
- Subset $L \subset V$ of labeled nodes.
- Length-$|L|$ vector $\boldsymbol{y}_L$ of outcomes on L (real-valued or categorical).

Problem output.

- Length-$|U|$ vector $\boldsymbol{y}_U$ of outcomes on $U = V \setminus L$ (real-valued or categorical).

We will discuss two broad classes of algorithms.

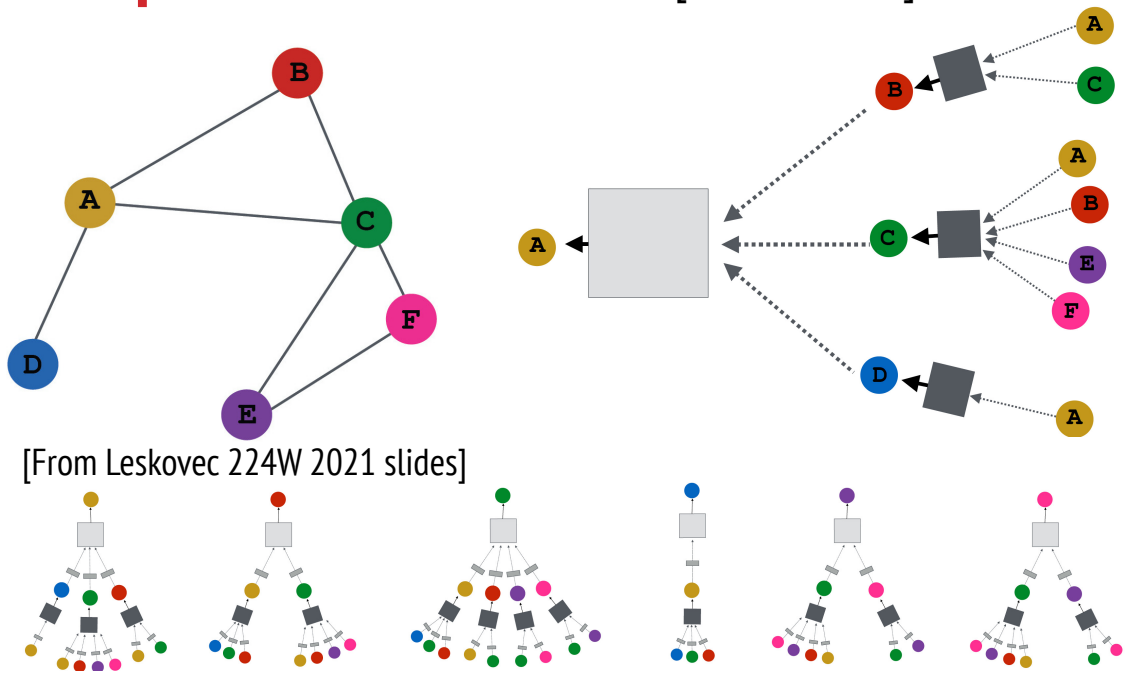
1. Label Propagation [early 2000s]



[From Gleich & Mahoney 2015]

- Propagate/spread/diffuse known values.
- Doesn't use features.

2. Graph Neural Networks [late 2010s]



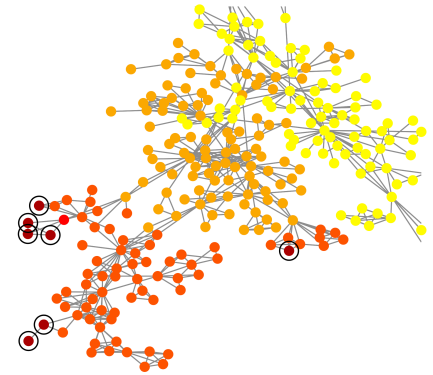
[From Leskovec 224W 2021 slides]

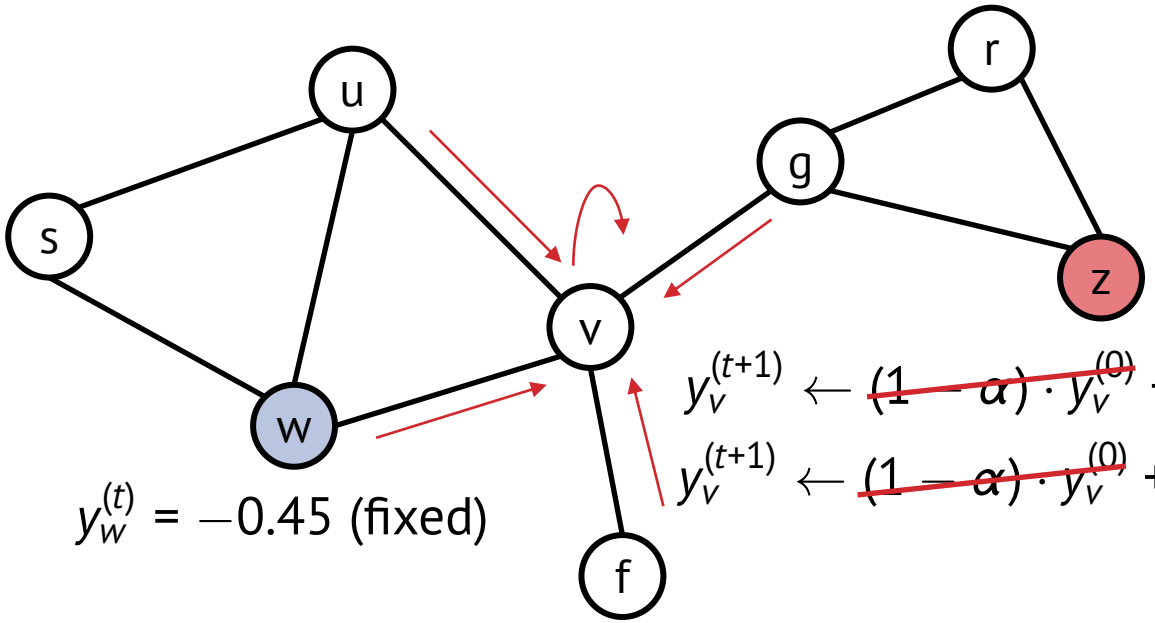
- Combine neighbor features via neural nets.
- Train with known outcomes.
- Produces vector $\boldsymbol{h}_v$ for each node *v*.

Key questions.

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

Label propagation is just neighbor averaging.





$y_z^{(t)} = 0.72$ (fixed)

$y_w^{(t)} = -0.45$ (fixed)

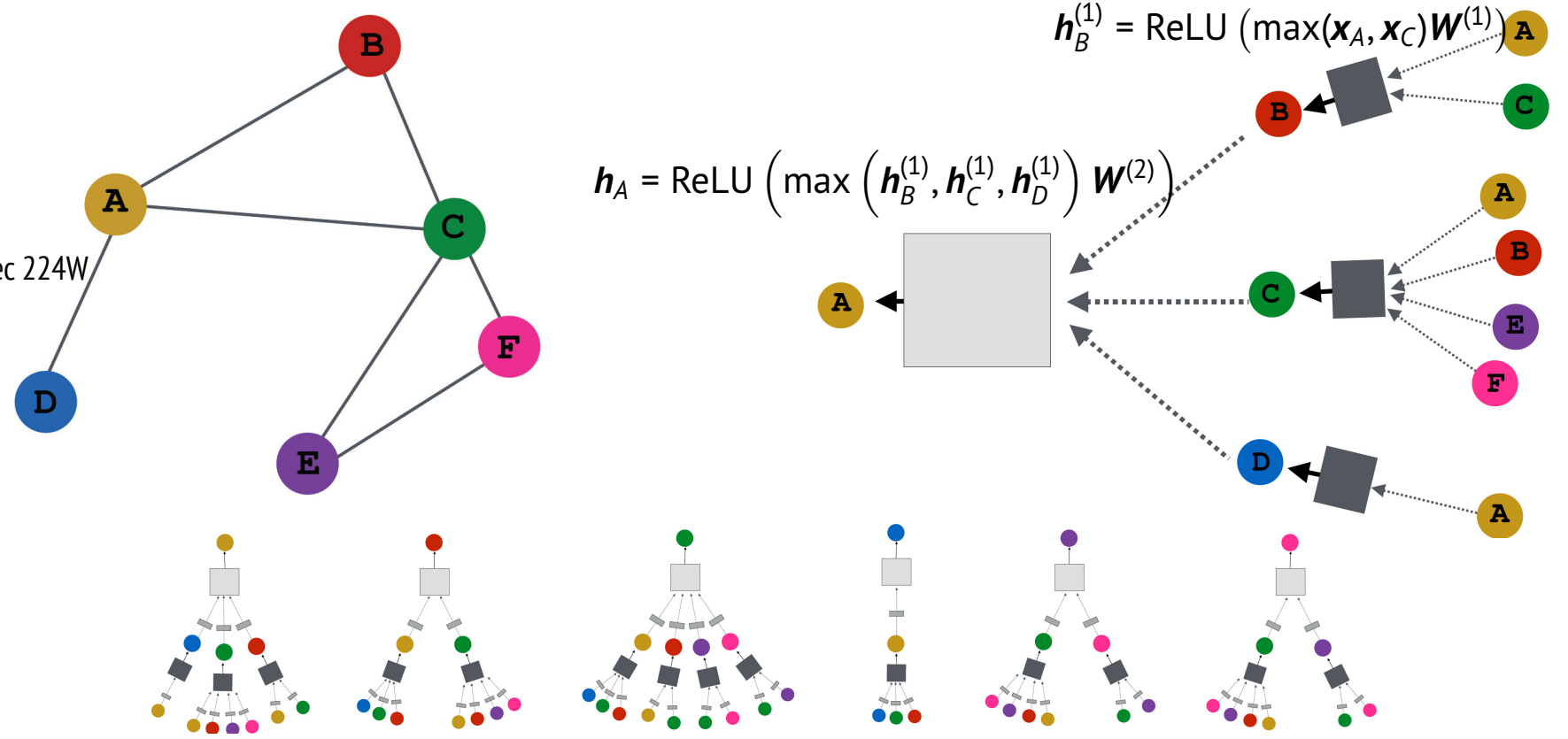
$$y_v^{(t+1)} \leftarrow \cancel{(1-\alpha) \cdot y_v^{(0)}} + \alpha \cdot \frac{1}{4}\left(y_u^{(t)} + y_w^{(t)} + y_f^{(t)} + y_g^{(t)}\right)$$

$$y_v^{(t+1)} \leftarrow \cancel{(1-\alpha) \cdot y_v^{(0)}} + \alpha \cdot \frac{1}{\sqrt{4}}\left(\frac{1}{\sqrt{3}}y_u^{(t)} + \frac{1}{\sqrt{3}}y_w^{(t)} + \frac{1}{\sqrt{1}}y_f^{(t)} + \frac{1}{\sqrt{3}}y_g^{(t)}\right)$$

- At convergence, everyone is roughly the average over their neighbors ⟶ smooth!
- **Regression.** Start with real values (0/mean at unknown) ⟶ smoothed value for each node.
- **Classification.** For each class, initial values 0 or 1 ⟶ score for each node for each class.

Graph neural networks aggregate features.

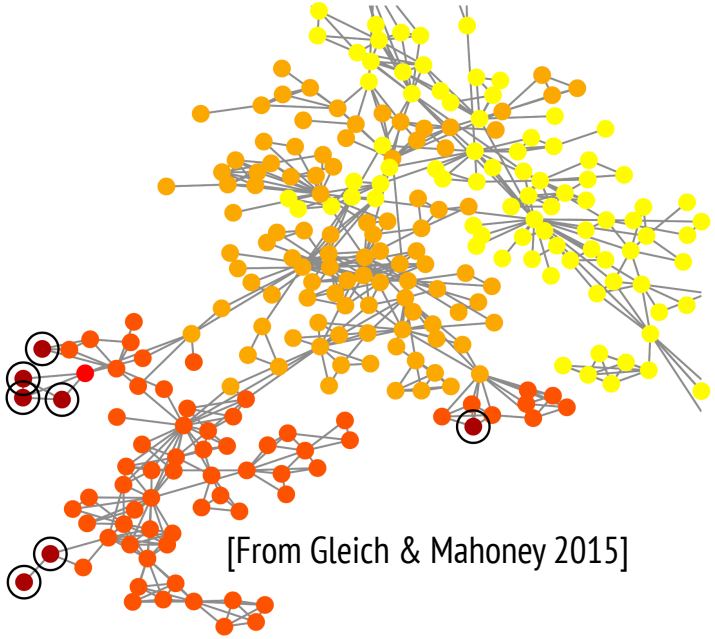
[From Leskovec 224W 2021 slides]



- **Regression.** Prediction at node A = $<\beta, h_A>$.
- **Classification.** Prob(node class $A = i$) proportional to $\exp(<\beta_i, h_A>)$.
- Big optimization problem trained with labeled nodes and automatic differentiation.

We will discuss two broad classes of algorithms.

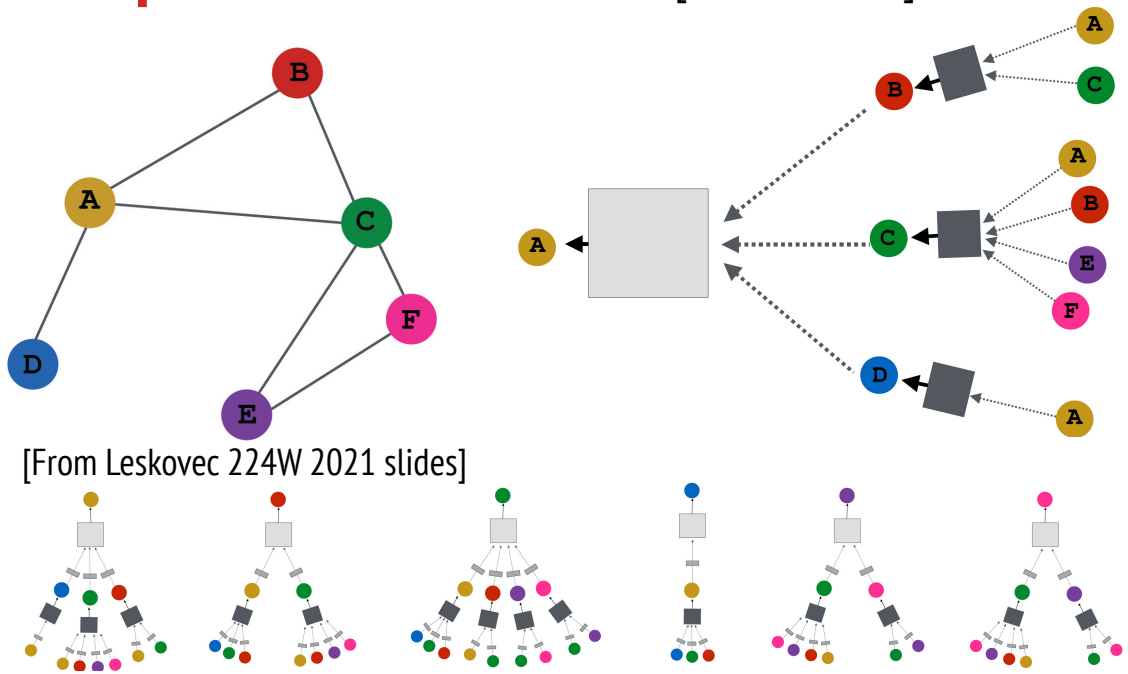
1. Label Propagation [early 2000s]



[From Gleich & Mahoney 2015]

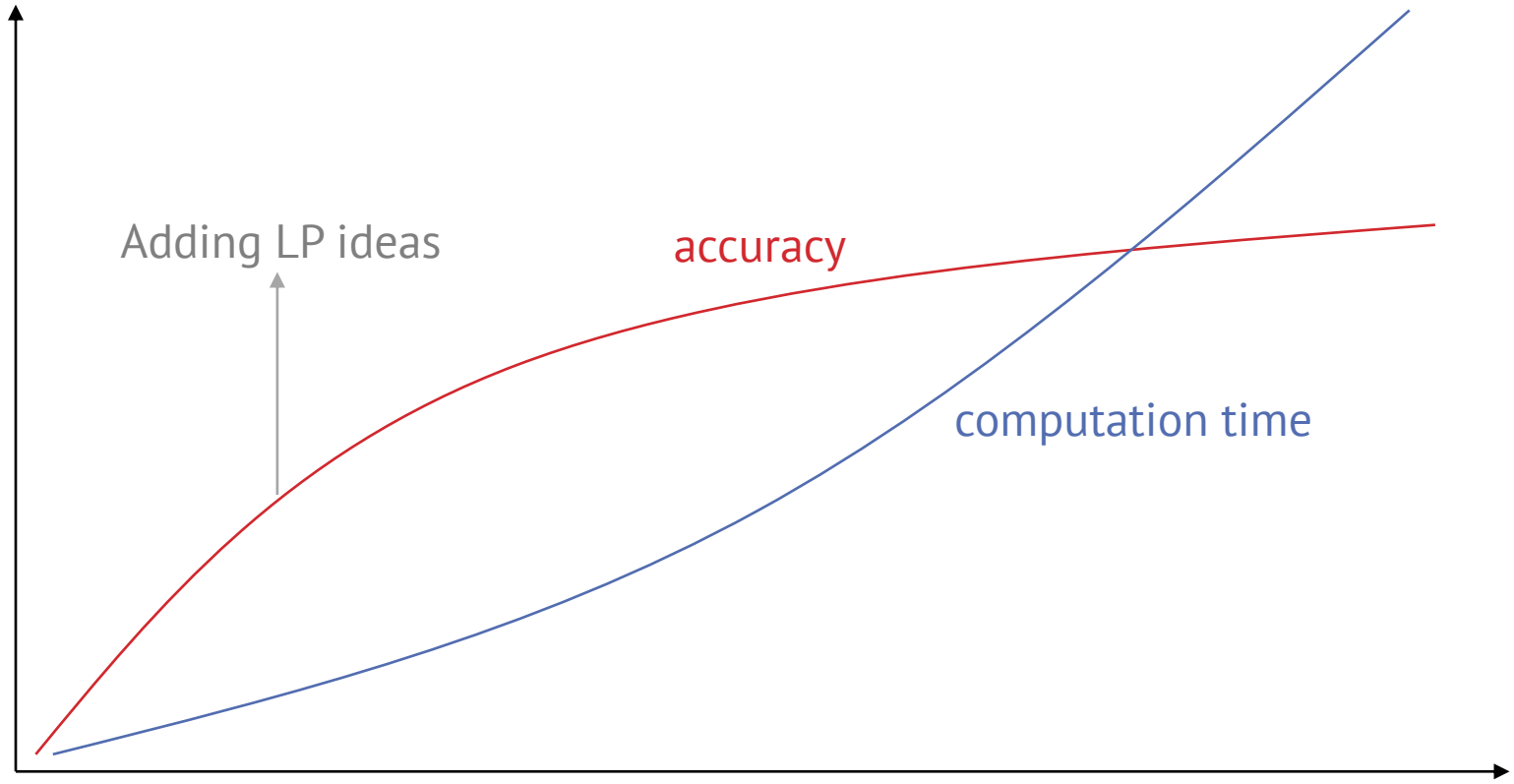
- Strong modeling assumption:
  connected nodes have similar labels.
- Works because of homophily [McPherson+ 01]
  a.k.a. assortativity [Newman 02]
- Why not use additional info/features?
- **FAST**
  a few sparse matrix-vector products

2. Graph Neural Networks [late 2010s]



[From Leskovec 224W 2021 slides]

- 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



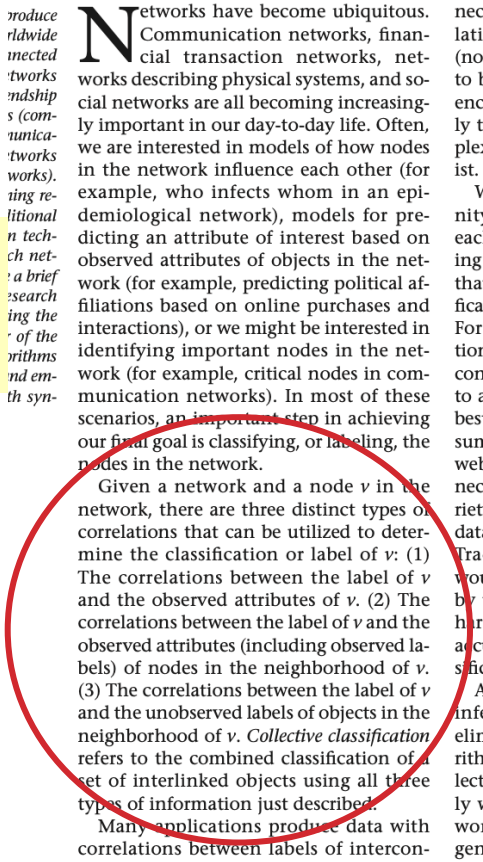
More use of node features ⟶
(bigger & fancier GNNs)

We are building on established ideas.

Collective Classification in Network Data

Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Gallagher, and Tina Eliassi-Rad

Given a network and a node v in the network, there are three distinct types of correlations that can be utilized to determine the classification or label of v: (1) The correlations between the label of v and the observed attributes of v. (2) The correlations between the label of v and the observed attributes (including observed labels) of nodes in the neighborhood of v. (3) The correlations between the label of v and the unobserved labels of objects in the neighborhood of v. *Collective classification* refers to the combined classification of a set of interlinked objects using all three types of information just described.



Networks have become ubiquitous. Communication networks, financial transaction networks, networks describing physical systems, and social networks are all becoming increasingly important in our day-to-day life. Often, we are interested in models of how nodes in the network influence each other (for example, who infects whom in an epidemiological network), models for predicting an attribute of interest based on observed attributes of objects in the network (for example, predicting political affiliations based on online purchases and interactions), or we might be interested in identifying important nodes in the network (for example, critical nodes in communication networks). In most of these scenarios, an important step in achieving our final goal is classifying, or labeling, the nodes in the network.

Given a network and a node v in the network, there are three distinct types of correlations that can be utilized to determine the classification or label of v: (1) The correlations between the label of v and the observed attributes of v. (2) The correlations between the label of v and the observed attributes (including observed labels) of nodes in the neighborhood of v. (3) The correlations between the label of v and the unobserved labels of objects in the neighborhood of v. *Collective classification* refers to the combined classification of a set of interlinked objects using all three types of information just described.

Many applications produce data with correlations between labels of interconnected nodes. The simplest types of correlation can be the result of homophily (nodes with similar labels are more likely to be linked) or the result of social influence (nodes that are linked are more likely to have similar labels), but more complex dependencies among labels often exist.

Within the machine-learning community, classification is typically done on each object independently, without taking into account any underlying network that connects the nodes. Collective classification does not fit well into this setting. For instance, in the web page classification problem where web pages are interconnected with hyperlinks and the task is to assign each web page with a label that best indicates its topic, it is common to assume that the labels on interconnected web pages are correlated. Such interconnections occur naturally in data from a variety of applications such as bibliographic data, email networks, and social networks. Traditional classification techniques would ignore the correlations represented by these interconnections and would be hard pressed to produce the classification accuracies possible using a collective classification approach.

Although traditional exact probabilistic inference algorithms such as variable elimination and the junction tree algorithm harbor the potential to perform collective classification, they are practical only when the graph structure of the network satisfies certain conditions. In general, exact inference is known to be

Copyright © 2008, Association for the Advancement of Artificial Intelligence. All rights reserved. ISSN 0738-4602

See also ReFeX (KDD 2011) and RolX (KDD 2012) for more prescient research by your instructor.

1. LP on residuals for improving GNN regression algorithms.
   Residual Correlation in Graph Neural Network Regression.
   Jia & Benson, KDD 2020.

2. Using LP but removing GNNs altogether for good classification performance.
   Combining Label Propagation and Simple Models Out-performs Graph Neural Networks.
   Huang et al., ICLR 2021.

3. Statistical framework that unifies LP and GNN ideas.
   A Unifying Generative Model for Graph Learning Algorithms: Label Propagation, Graph Convolutions, and Combinations.
   Jia & Benson, arXiv 2021.

The formal problem we are solving.

Problem input.

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

Problem output.

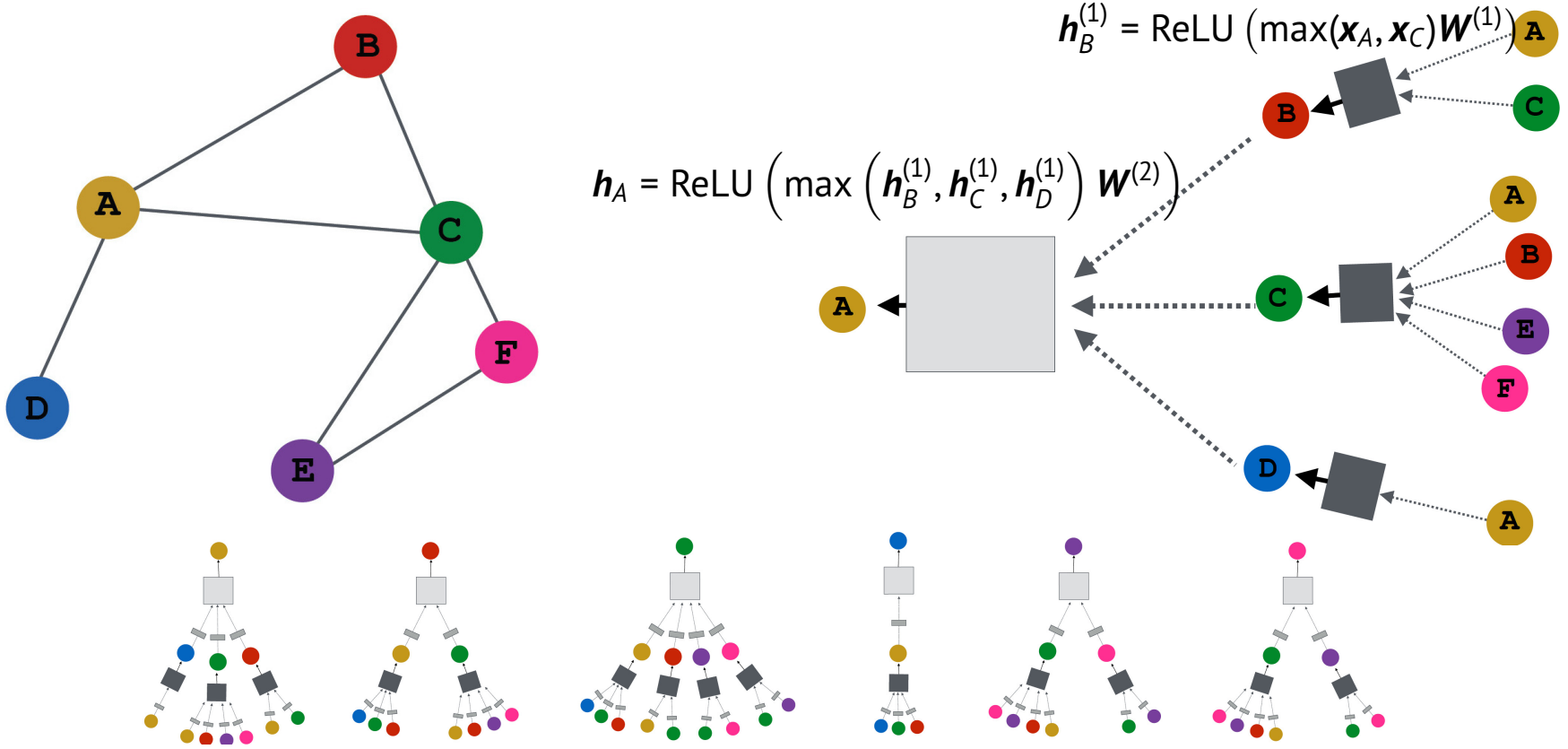
- Length-$|U|$ vector $\boldsymbol{y}_U$ of real-valued outcomes on $U = V \setminus L$.

Solution evaluation.

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

$$R^2 = 1 - \frac{\sum_{u \in U} [y_u^* - y_u]^2}{\sum_{u \in U} [y_u^* - \bar{y}_u^*]^2}, \quad y_u^* = \text{true value at } u$$

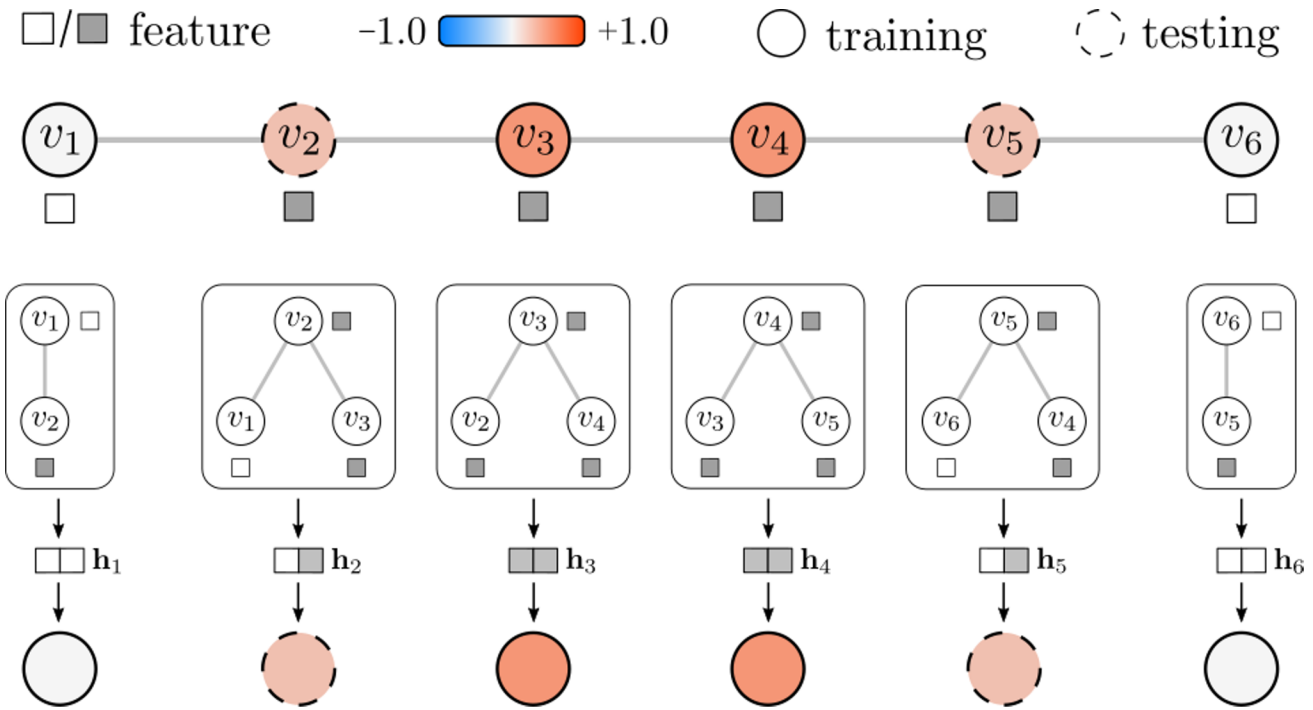
Graph neural networks make uncorrelated predictions.



- Use labels to find representation vectors h_A, h_B, h_C, h_D, h_E, and h_F and coefficients β.
- Given representations and coefficients, predictions are independent.
- Something strange? Compared to LP, use of labels is very implicit.
- Pervasive paradigm [Kipf-Welling 16; Hamilton+ 17; Zhou+ 18; ~10,000 papers in 5 years]

Graph neural networks make uncorrelated predictions.

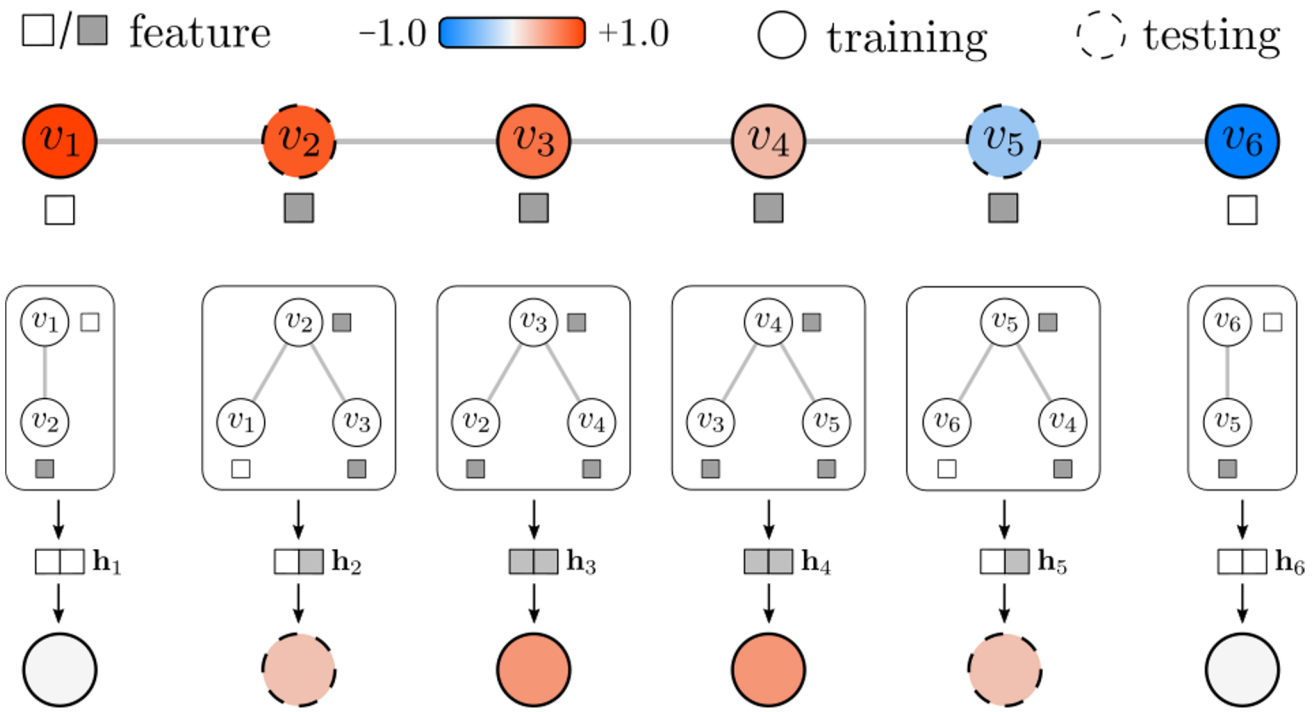
Color is true label



1. Form local neighborhoods
2. Combine features to get a representation $\boldsymbol{h}_v$ at node v.
3. Predict outcome given representation (learn model params w/ training data)

- If node features are overwhelmingly predictive, these uncorrelated predictions might be OK.

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

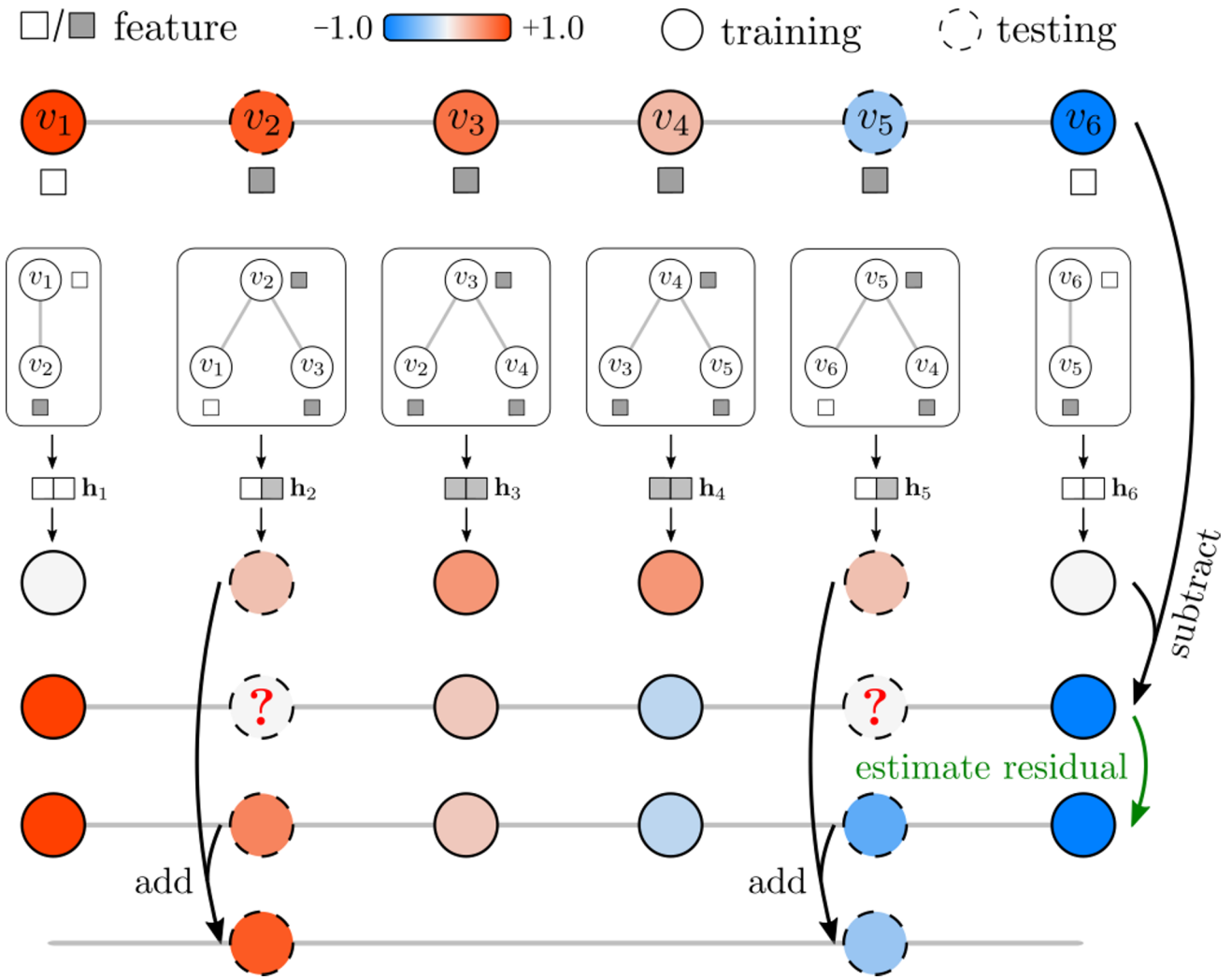


New labels…

… but same predictions

- 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.



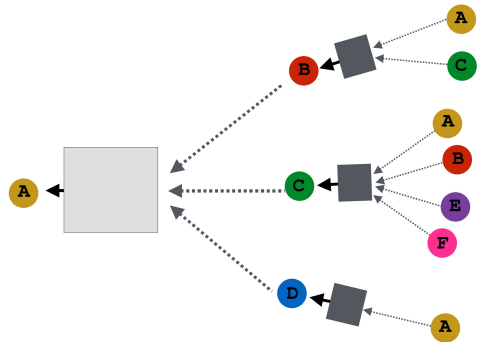
Works with any GNN. Just layer on top.

1. Standard GNN prediction.
2. Compute residual error.
3. Propagate residuals to estimate errors on test nodes.
4. Add residual to base prediction.

The residual propagation algorithm is simple.

1. Make a base prediction on each node with any method.
2. residual = true value – base prediction (on labeled nodes)
3. Label propagation on residual ⟶ smooth errors
4. Final prediction = smoothed residual + base prediction (= true value on labeled nodes)

There is a simple statistical motivation for why residual propagation works.



The objective looks like ordinary least squares (OLS).

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

- 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).
- Error $\varepsilon_u = y_u - \beta^T \mathbf{h}_u$ = residual at u.
- We shouldn't expect i.i.d. error in graph data!

- We are positing that errors are positively correlated along edges $\mathbf{r} = \boldsymbol{\varepsilon} \sim N(\mathbf{0}, (\mathbf{I} - \alpha \mathbf{S})^{-1})$

- Essentially, this is a type of generalized least squares or kriging [Aitken 36; Xu-Dyer-Owen 10; Chin+ 19].

A little background on multivariate Gaussians.

$\boldsymbol{r} = \varepsilon \sim N(\mathbf{0}, (\boldsymbol{I} - \alpha \boldsymbol{S})^{-1}) = N(\mathbf{0}, \boldsymbol{\Gamma}^{-1}) = N(\mathbf{0}, \Sigma)$, so $\boldsymbol{\Sigma}^{-1} = \boldsymbol{\Gamma} = \boldsymbol{I} - \alpha \boldsymbol{S}$

Facts.

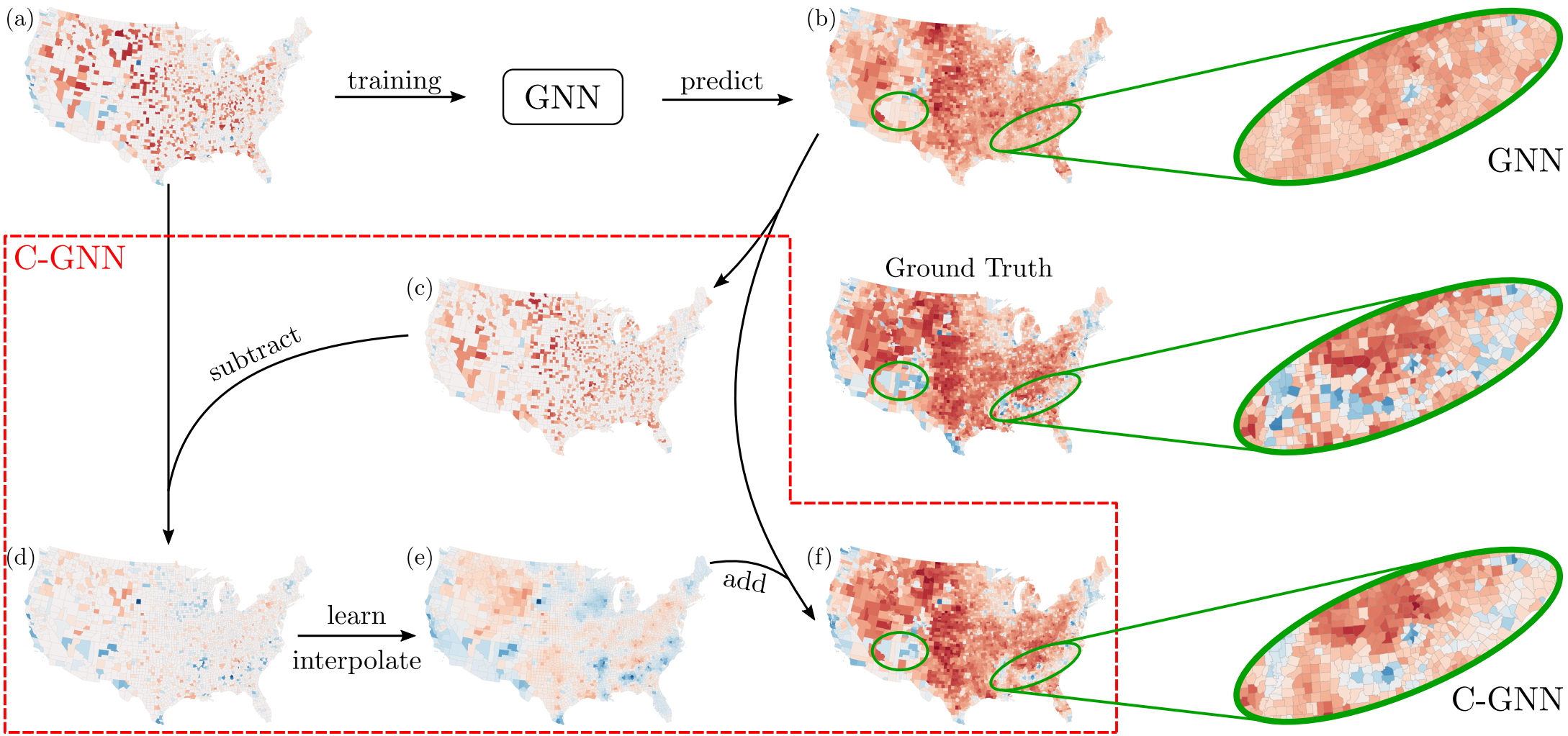
1. $\boldsymbol{\Sigma}^{-1}_{ij} = \boldsymbol{\Gamma}_{ij} = 0 \to r_i, r_j$ independent given other entries in $\boldsymbol{r}$
   (so $\boldsymbol{\Sigma}^{-1}_{ij} = \boldsymbol{\Gamma}_{ij} = 0 \iff (i,j)$ not an edge)

2. $\boldsymbol{r}_U | \boldsymbol{r}_L \sim N(-\boldsymbol{\Gamma}_{UU}^{-1} \boldsymbol{\Gamma}_{UL} \boldsymbol{r}_L, \boldsymbol{\Gamma}_{UU}^{-1})$

3. Mean minimizes mean squared (so we use $-\boldsymbol{\Gamma}_{UU}^{-1} \boldsymbol{\Gamma}_{UL} \boldsymbol{r}_L$)

4. $-\boldsymbol{\Gamma}_{UU}^{-1} \boldsymbol{\Gamma}_{UL} \boldsymbol{r}_L = -(I - \alpha \boldsymbol{S}_{UU})^{-1} (\alpha \boldsymbol{S}_{UL}) \boldsymbol{r}_L$, which is the limit of LP:
   $r_i^{(t+1)} \leftarrow \alpha \cdot \frac{1}{\sqrt{d_u}} \sum_{(i,j) \in E} \frac{r_j^t}{\sqrt{d_j}}$.

Residual propagation works well in practice.

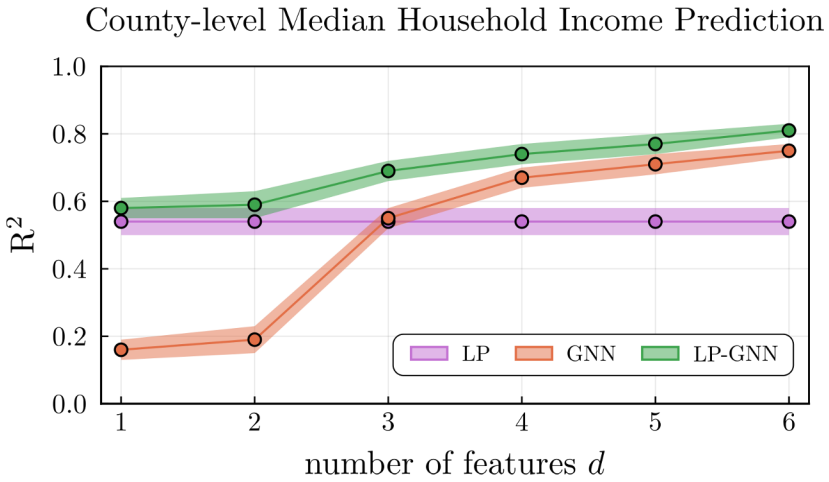


Out-of-sample R^2 0.51 ⟶ 0.69.

Residual propagation works well in practice.

R^2 on county-level demographics predictions.

vote share	$0.51 \rightarrow 0.69$
income	$0.75 \rightarrow 0.81$
education level	$0.70 \rightarrow 0.72$
unemployment level	$0.55 \rightarrow 0.75$



R^2 on traffic predictions.

Anaheim	$0.76 \rightarrow 0.81$
Chicago	$0.68 \rightarrow 0.72$

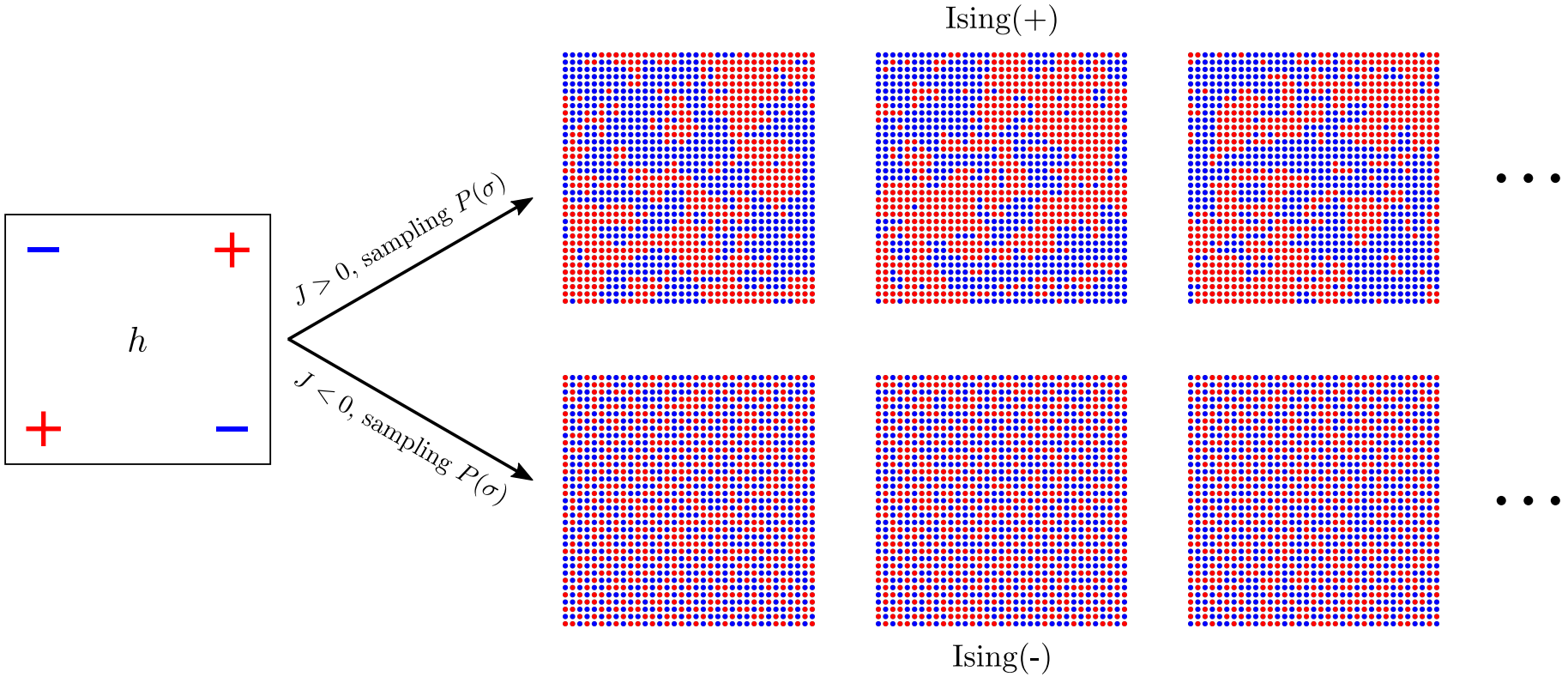
We can also learn the correlation directly.

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

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

- Jointly maximize likelihood of base model + correlated error.
- Requires lots of numerical tricks to be scalable.
- Some empirical performance boosts.

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

GNNs and label propagation can be combined.

1. **Key idea.** Run LP on the *residuals* for *correlated errors*.
2. Residual propagation is just a post-processing step that can be used with any regression method (we never see it hurt)
3. When features are somewhat but not overwhelmingly predictive, residual propagation can boost performance substantially.

1. LP on residuals for improving GNN regression algorithms.
   Residual Correlation in Graph Neural Network Regression.
   Jia & Benson, KDD 2020.

2. Using LP but removing GNNs altogether for good classification performance.
   Combining Label Propagation and Simple Models Out-performs Graph Neural Networks.
   Huang et al., ICLR 2021.

3. Statistical framework that unifies LP and GNN ideas.
   A Unifying Generative Model for Graph Learning Algorithms: Label Propagation, Graph Convolutions, and Combinations.
   Jia & Benson, arXiv 2021.

Can we get top performance without GNNs?

OGB

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: >=1.1.1

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

The formal problem we are solving.

Problem input.

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

Problem output.

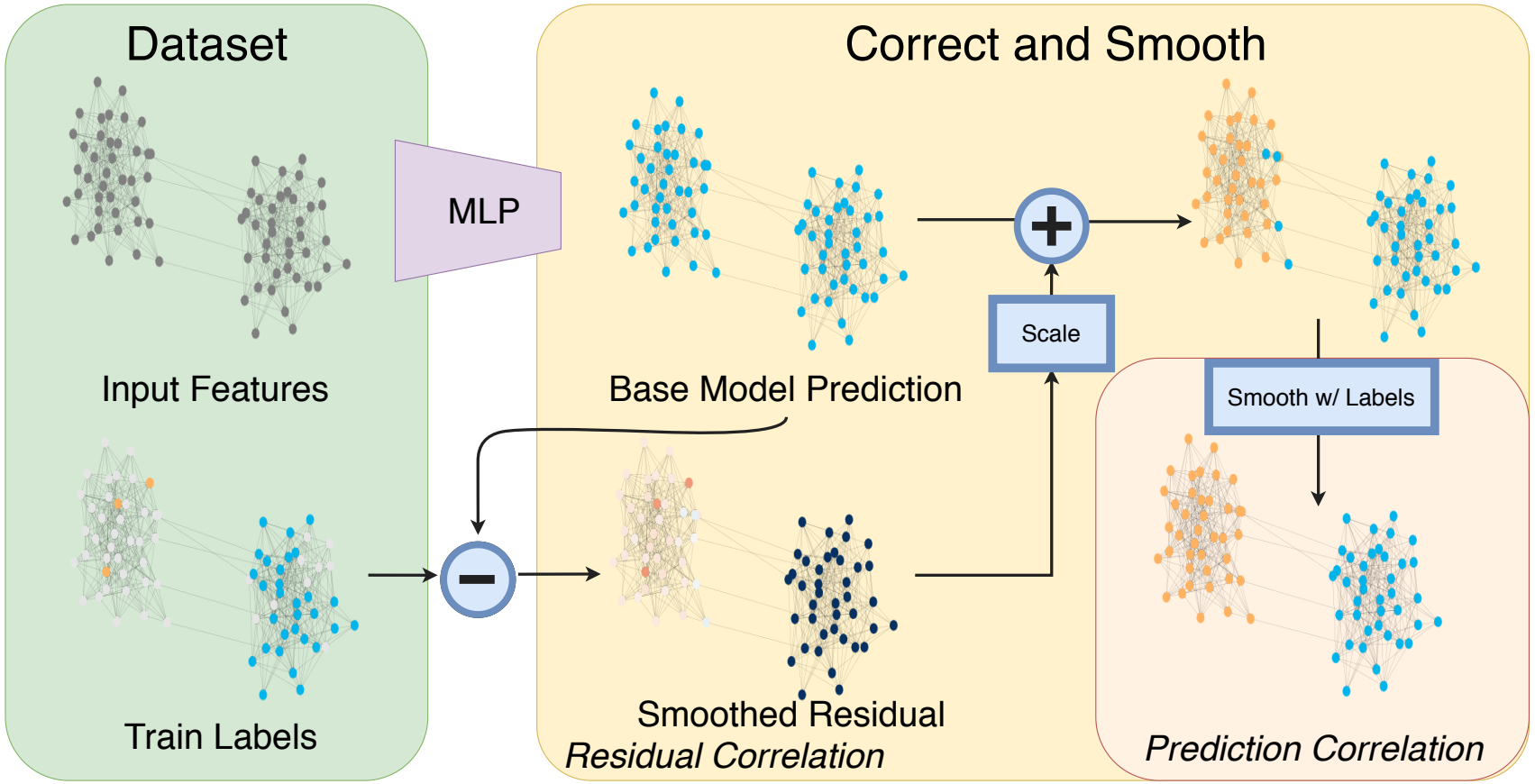
- Length-$|U|$ vector $\boldsymbol{y}_U$ of categorical outcomes (classes) on $U = V \setminus L$.

Solution evaluation.

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

We can get great classification accuracy without GNNs.

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

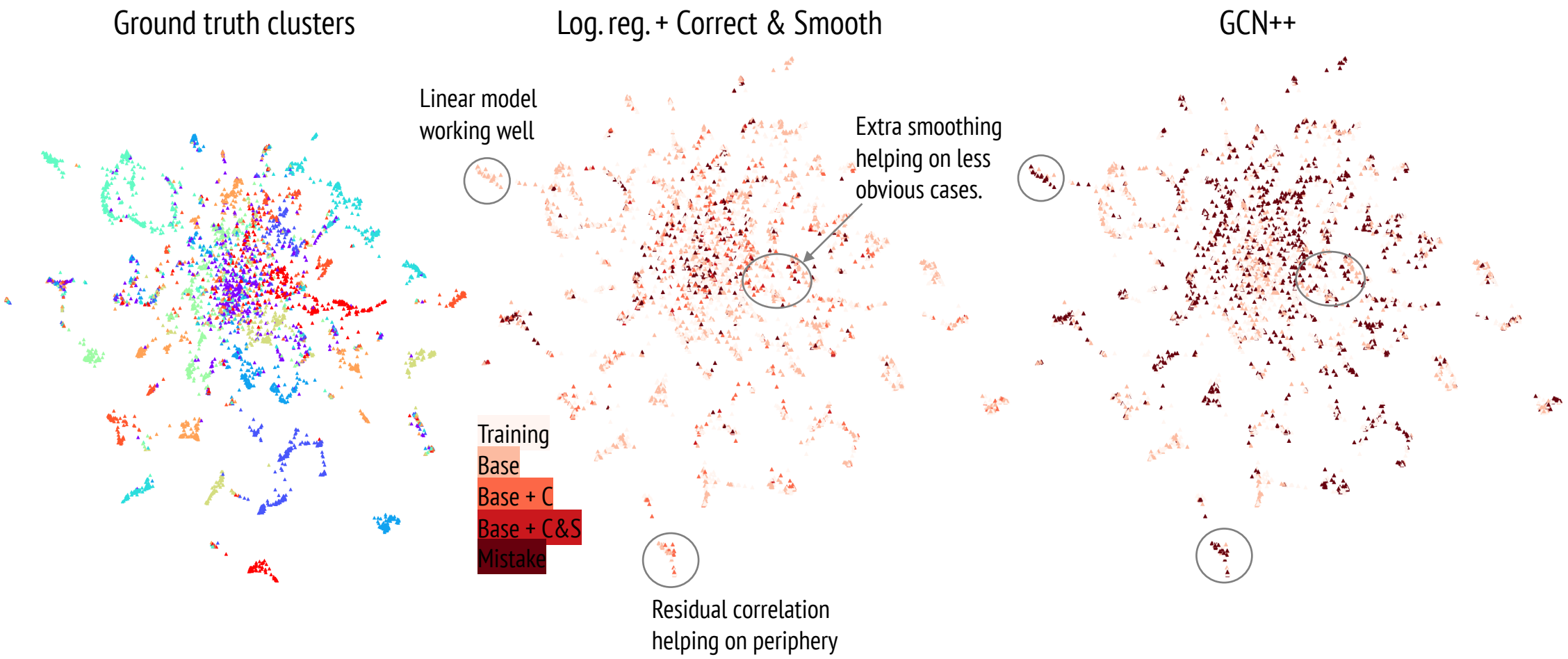


We use base predictions that ignore the graph structure and treat the nodes independently.

- Prob(node class $A = i$) proportional to $\exp(<\boldsymbol{\beta}_i, \boldsymbol{h}_A>)$.
- Logistic regression: $\boldsymbol{h}_A = \boldsymbol{x}_A$
- Multilayer perceptron: $\boldsymbol{h}_A = \text{ReLU}(\boldsymbol{W}\,\text{ReLU}(\boldsymbol{W}\boldsymbol{x}_A))$.
- Much faster to train than GNN.

We get a lot of mileage out of smoothness.

1. **Base predictions.**
   Base predictor ⟶ a vector of class probabilities $\boldsymbol{p}_u$ at each node u.
2. **Residual propogation adaptation (correction step).**
   Form error vector $\boldsymbol{e}_u$ = one_hot($\boldsymbol{y}_u$) – $\boldsymbol{p}_u$ at labeled nodes.
   Run residual prop on each component ⟶ smoothing residual vector $\boldsymbol{r}_v$.
   Scaling not quite right, correct with $\boldsymbol{z}_v = \boldsymbol{p}_v + s * \boldsymbol{r}_v$ (tune scalar s).
3. **Reset on labeled nodes.**
   $\boldsymbol{z}_u$ = one_hot(y_u) if u labeled; $\boldsymbol{z}_v$ same if v unlabeled (not a probability, though).
4. **Smooth corrected vectors (smooth step).**
   Run LP on each coordinate of z to get smoothed vectors $\boldsymbol{y}_v$.
5. **Final predictions.**
   For unlabeled node v, predict maximum entry in $\boldsymbol{y}_v$.



Predicting dorm residence with Rice University Facebook friendship network from 2005.

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

LP + simple models can get top performance.

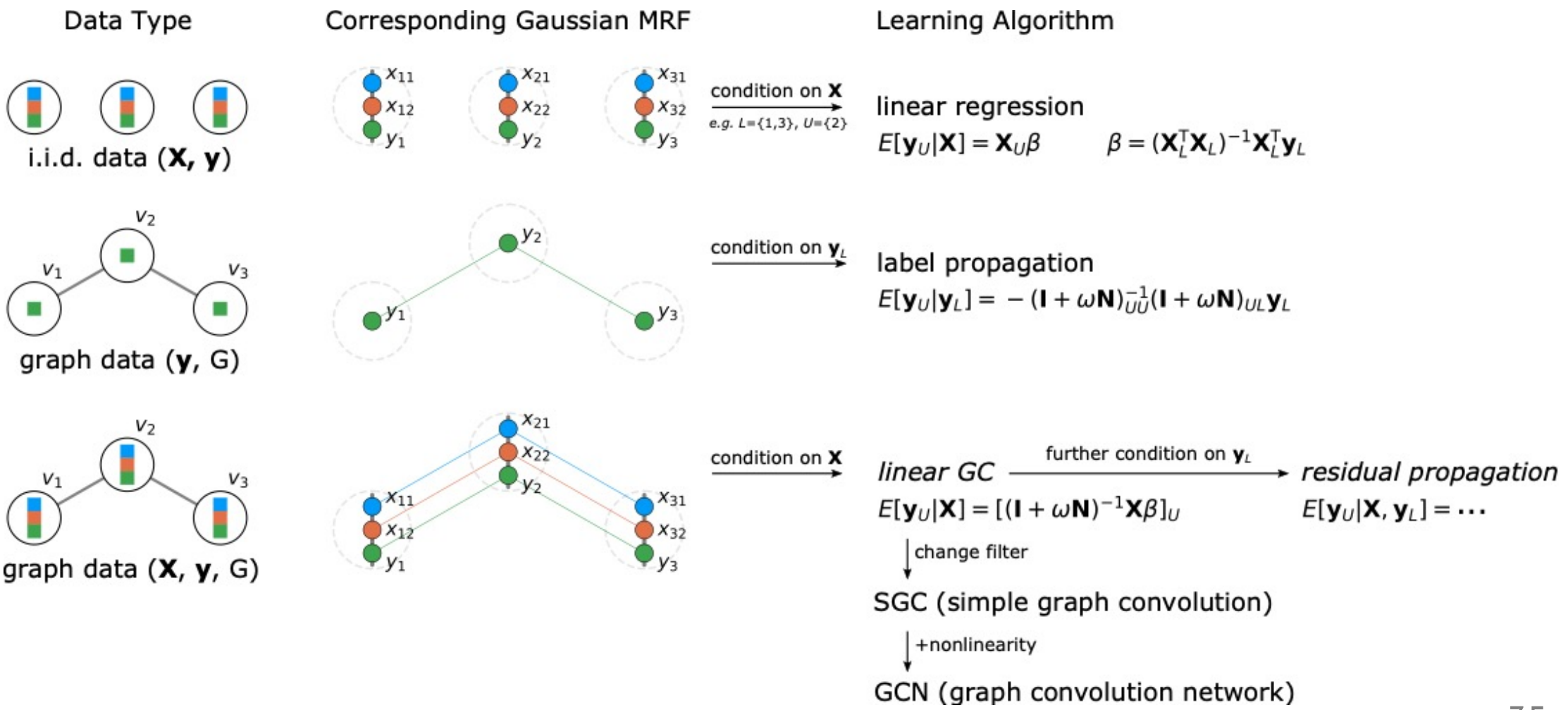
1. Avoiding GNNs speeds up training substantially.
2. Smoothing errors and final predictions help performance.
3. We also get some benefits by “feature augmentation” such as spectral embedding (could also use motif counts, community memberships, …)
4. Just principled heuristics at this point.

1. LP on residuals for improving GNN regression algorithms.
   Residual Correlation in Graph Neural Network Regression.
   Jia & Benson, KDD 2020.

2. Using LP but removing GNNs altogether for good classification performance.
   Combining Label Propagation and Simple Models Out-performs Graph Neural Networks.
   Huang et al., ICLR 2021.

3. Statistical framework that unifies LP and GNN ideas.
   A Unifying Generative Model for Graph Learning Algorithms: Label Propagation, Graph Convolutions, and Combinations.
   Jia & Benson, arXiv 2021.

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



Our model is based on smooth random attributes.

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

$$\phi(\boldsymbol{A}|\boldsymbol{H},\boldsymbol{h}) = \frac{1}{2}\sum_{u=1}^{n} \boldsymbol{a}_u^\mathsf{T}\boldsymbol{H}\boldsymbol{a}_u + \frac{1}{2}\sum_{i=1}^{p+1} h_i \boldsymbol{A}_i^\mathsf{T}\boldsymbol{N}\boldsymbol{A}_i, \quad \boldsymbol{H} \in \mathbb{R}^{(p+1)\times(p+1)} \text{ spd}, \quad \boldsymbol{0} \le \boldsymbol{h} \in \mathbb{R}^{(p+1)}$$

correlated attributes on a node

smoothness on attributes

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

$$\rho(\boldsymbol{A} = \boldsymbol{A}|\boldsymbol{H},\boldsymbol{h}) = \frac{e^{-\phi(\boldsymbol{A}|\boldsymbol{H},\boldsymbol{h})}}{\int d\boldsymbol{A}'\ e^{-\phi(\boldsymbol{A}'|\boldsymbol{H},\boldsymbol{h})}}$$

Smoother attributes are more likely (homophily / assortativity)

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

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

Problem output.

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

Problem solution.

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

Case 1. Linear regression when there are no edges.

(special case of standard theory of linear models)

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

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

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

rather than infer directly,
estimate with OLS

$$\min_{\beta} \sum_{u\in L}[\boldsymbol{x}_u^T\beta - y_u]^2 = \min_{\beta} \|\boldsymbol{X}_U\beta - \boldsymbol{y}_U\|_2^2$$

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

One attribute, so these are just positive scalars

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

control noise, 1/H is variance if no edges

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

control smoothness

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

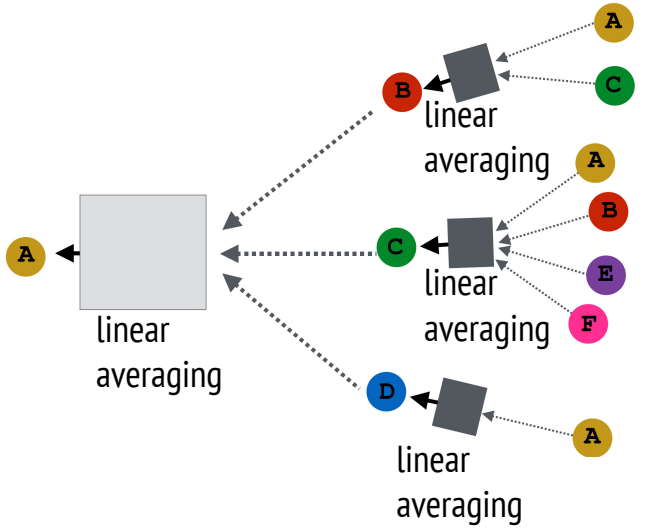
limit of label prop!

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

$$\alpha = {}^{\omega}/_{1+\omega} = {}^{h/H}/_{1+h/H} \in (0, 1)$$

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

Case 3. Linear graph convolutions when conditioning on features.



control noise (inverse variance)

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

control smoothness

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

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

LP on *features*! (no unknown values, though)

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

$$\forall v \in V,\ x_v^{(0)} = x_v$$

Linear graph convolution (LGC).

1. Run LP on each feature ⟶ smoothed features.
2. Ordinary least squares on these preprocessed, smoothed features.

Case 3. Linear graph convolutions when conditioning on features.

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

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

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

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

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

Similarly, GNN predictions should not be independent given representations!

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

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

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

LP on *residual errors!*

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

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

Can substitute in any initial prediction.

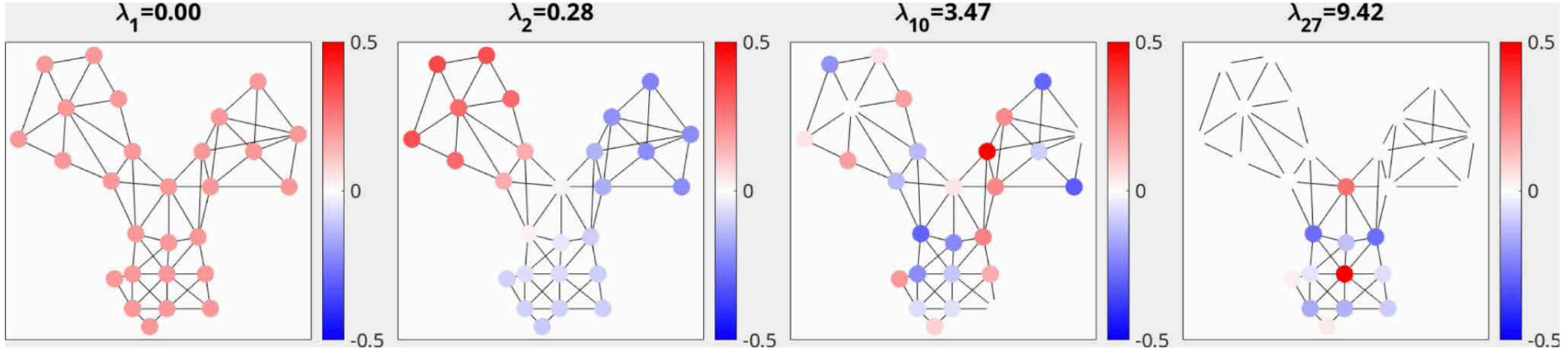
Dataset	Outcome	LP	LR	LGC (α)	SGC (K)	GCN (K)	LGC/RP	SGC/RP	GCN/RP
U.S.	income	0.40	0.63	0.66 (0.46)	0.51 (1.0)	0.53 (1.3)	**0.69**	0.55	0.55
	education	0.31	**0.71**	**0.71** (0.00)	0.43 (1.0)	0.47 (1.0)	**0.71**	0.46	0.48
	unemployment	0.47	0.34	0.39 (0.59)	0.32 (1.3)	0.45 (2.5)	**0.54**	0.52	0.53
	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 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 $\longrightarrow$ larger K, α
- Adding residual prop never hurts!
- GCN better with more homophily?
  "memorizing" neighborhood features (zero training error)
  + smoothness in data $\longrightarrow$ better out-of-sample prediction

Our model helps us understand smoothing.



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

$$\boldsymbol{N} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^{\mathsf{T}}, \text{ feature } \boldsymbol{f} = \sum_{i=1}^{n} c_i v_i$$

LGC $$\boldsymbol{f} \rightarrow \sum_{i=1}^{n} \frac{1}{(1+\omega\lambda_i)} c_i v_i$$

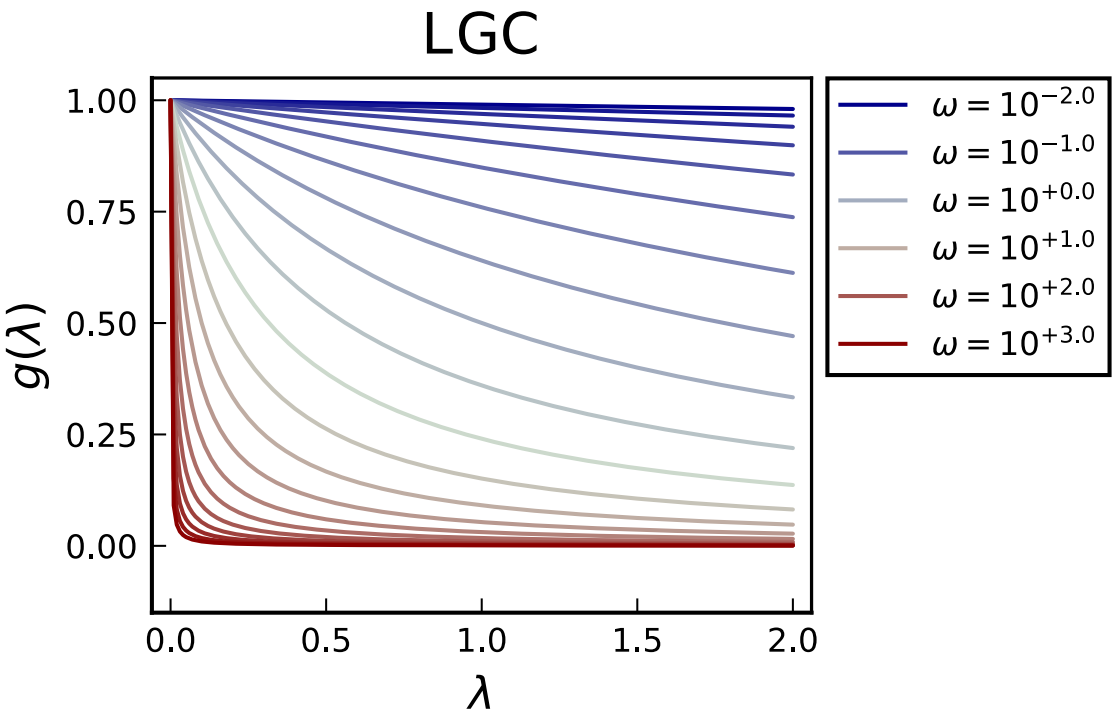
Low-pass on $[0, \infty)$, continuous parameterization.

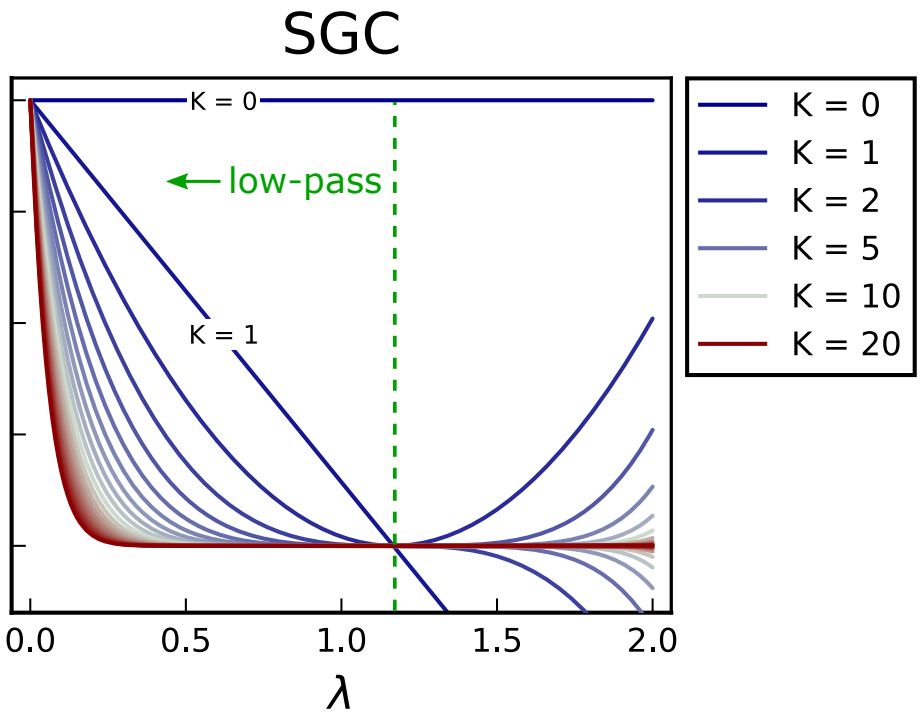
SGC $$\boldsymbol{f} \rightarrow \sum_{i=1}^{n} (1 - {}^{d}/_{(d+1)}\lambda_i)^{K} c_i v_i$$

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

Encouraging smoothness.

Our model helps us understand smoothing.





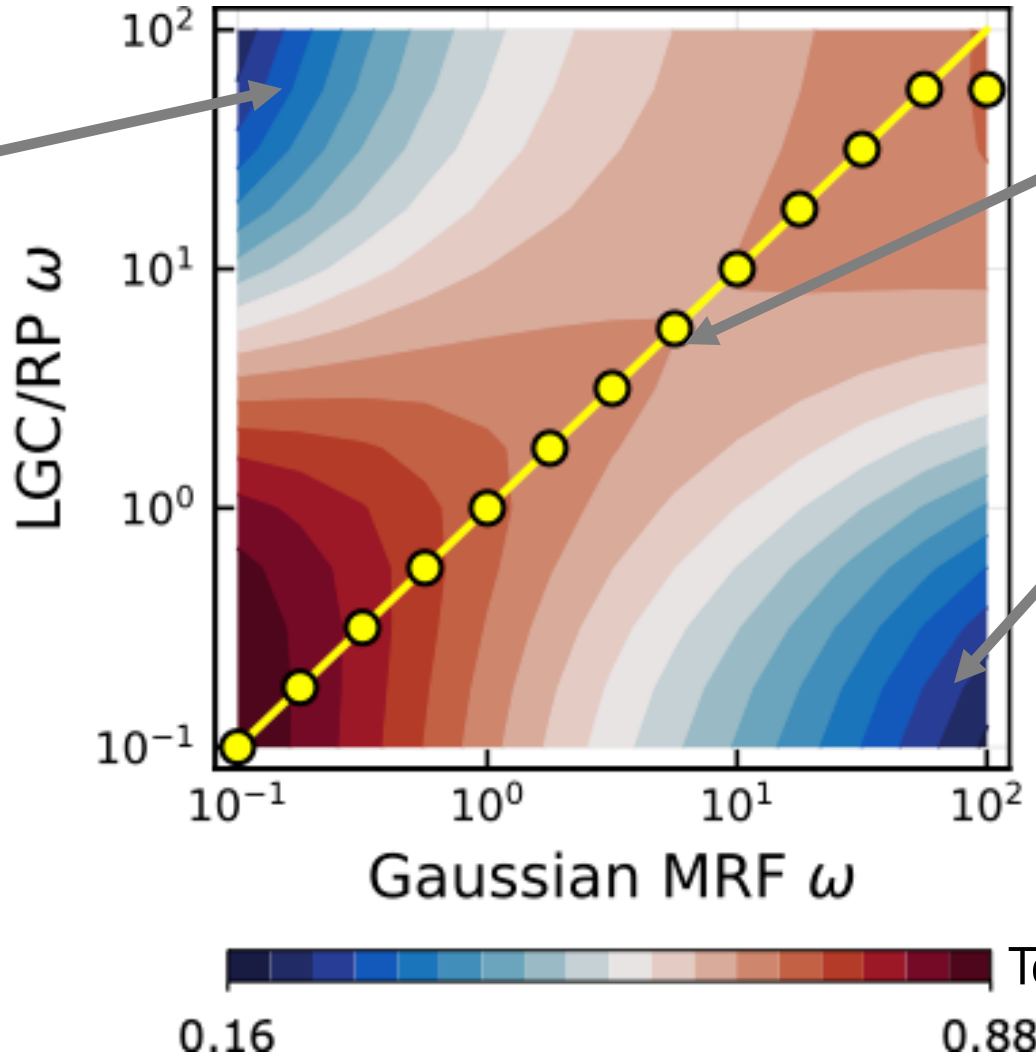
$$\boldsymbol{f} = \sum_{i=1}^{n} c_i v_i \rightarrow \sum_{i=1}^{n} \frac{1}{(1+\omega\lambda_i)} c_i v_i$$

$$\boldsymbol{f} = \sum_{i=1}^{n} c_i v_i \rightarrow \sum_{i=1}^{n} (1 - {}^{d}/_{(d+1)}\lambda_i)^{K} c_i v_i$$

Our model helps us understand smoothing.

"Oversmoothing."

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



cross validation can identify the model parameter.

Undersmoothing?

Possible but not discussed in the literature.

Major takeaway. Label propagation is a powerful tool.

1. LP can be applied to residuals (correlated errors).
2. LP can be applied to features (smoothing / de-noising).
3. LP can be applied to final predictions (more smoothness).
4. While traditionally seen as separate ideas, LP and basic GNN ideas can be derived from a common model and combined effectively.
5. Linear models are often superior to nonlinear ones (GNNs) in practice.

$$\boldsymbol{y}_U^{\text{LGC/RP}} = [(\boldsymbol{I}_n + \omega \boldsymbol{N})^{-1} \boldsymbol{X} \beta]_U - (\boldsymbol{I} + \omega \boldsymbol{N})_{UU}^{-1} (\boldsymbol{I} + \omega \boldsymbol{N})_{UL} \left(\boldsymbol{y}_L - [(\boldsymbol{I}_n + \omega \boldsymbol{N})^{-1} \boldsymbol{X} \beta]_L \right)$$

There are lots of open research directions.

1. Theory or more principled approaches for classification?

2. Similar ideas for other graph problems?
   link prediction, random walk prediction, graph classification, …

3. Generative models to explain other GNN ideas?
   attention, GraphSAGE, skip connections, …

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, 2021.
https://github.com/CUAI/CorrectAndSmooth

Supported by ARO MURI, ARO Award W911NF19- 1-0057, NSF Award DMS-1830274, JP Morgan Chase & Co, and Cornell University.

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?

Our model provides a nice setup for inductive learning.

Problem input.

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

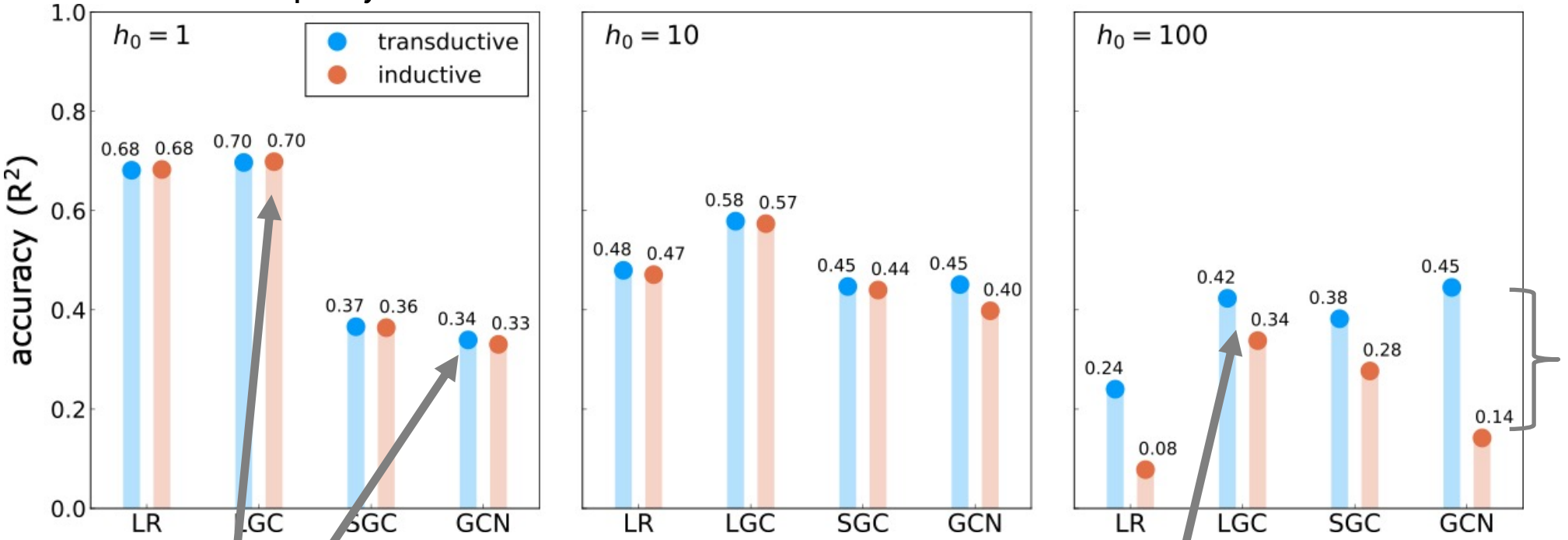
Problem output.

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

Our model provides a nice setup for inductive learning.

Predictive features,
low homophily.

High homophily.

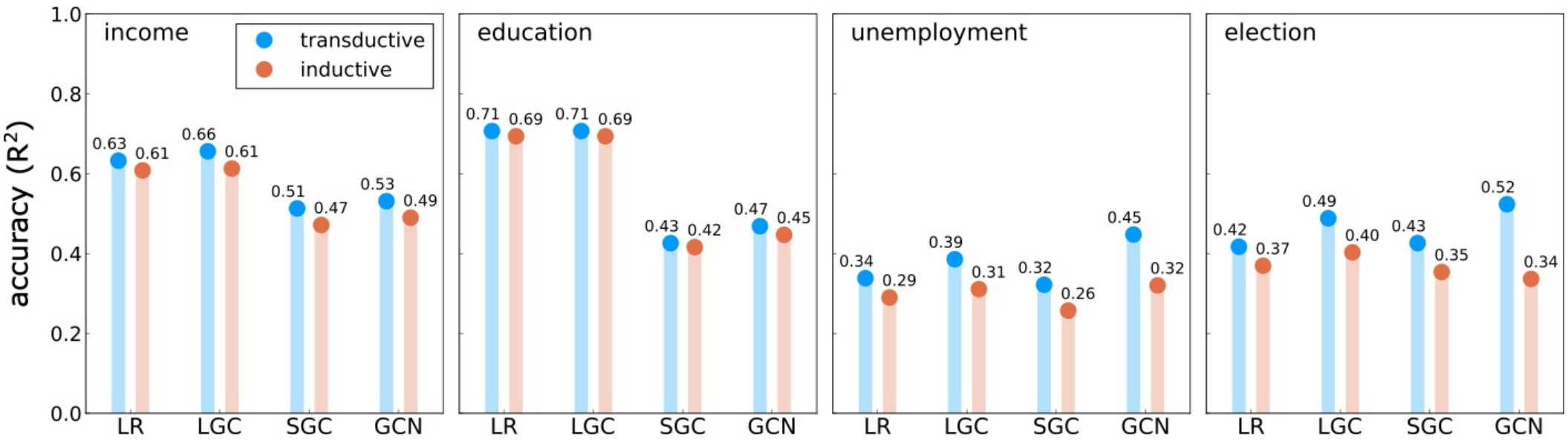


Bad overfitting!

No performance degradation.

A bit of degradation.

Our model provides a nice setup for inductive learning.



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