Bipartite Edge Prediction via Transductive Learning over Product Graphs

Hanxiao Liu, Yiming Yang

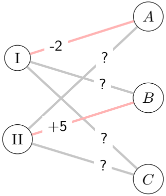
School of Computer Science, Carnegie Mellon University

July 8, 2015

Outline

Problem Description

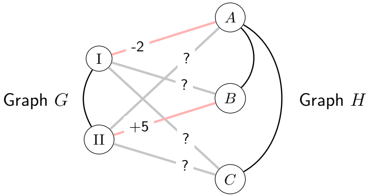
- Many applications involve predicting the edges of a bipartite graph.



1. Recommender System
2. Host-Pathogen Interaction
3. Question-Answering Mapping
4. Citation Network . . .

Problem Description

- Many applications involve predicting the edges of a bipartite graph.

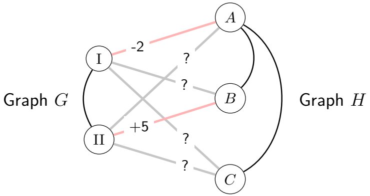


1. Recommender System
2. Host-Pathogen Interaction
3. Question-Answering Mapping
4. Citation Network ...

- Sometimes, vertex sets on both sides are intrinsically structured.

Problem Description

- Many applications involve predicting the edges of a bipartite graph.

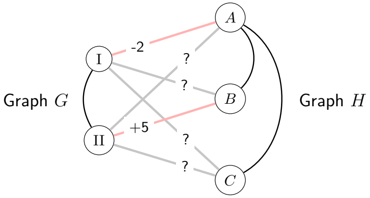


1. Recommender System
2. Host-Pathogen Interaction
3. Question-Answering Mapping
4. Citation Network ...

- Sometimes, vertex sets on both sides are intrinsically structured.
- Heterogeneous info: G + H + partial observations

Problem Description

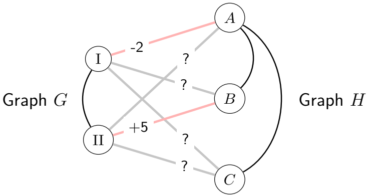
- Many applications involve predicting the edges of a bipartite graph.



1. Recommender System
2. Host-Pathogen Interaction
3. Question-Answering Mapping
4. Citation Network . . .

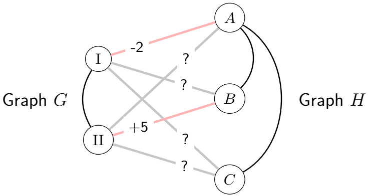
- Sometimes, vertex sets on both sides are intrinsically structured.
- Heterogeneous info: $G + H +$ partial observations
- Combine them to make better edge predictions?

The Proposed Framework



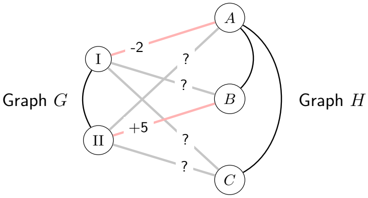
- Transductive learning should be effective
  1. Labeled edges (red) are highly sparse
  2. Unlabeled edges (gray) are massively available

The Proposed Framework



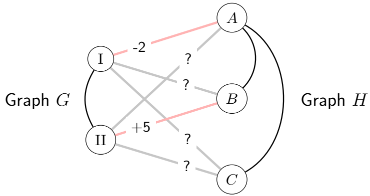
- Transductive learning should be effective
  1. Labeled edges (red) are highly sparse
  2. Unlabeled edges (gray) are massively available
- Assumption: similar edges should have similar labels

The Proposed Framework



- Transductive learning should be effective
  1. Labeled edges (red) are highly sparse
  2. Unlabeled edges (gray) are massively available
- Assumption: similar edges should have similar labels
- Prerequisite: a similarity measure among the edges, i.e. a "Graph of Edges" (not directly provided)

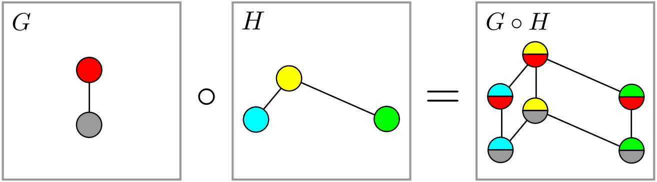
The Proposed Framework



- Transductive learning should be effective
  1. Labeled edges (red) are highly sparse
  2. Unlabeled edges (gray) are massively available
- Assumption: similar edges should have similar labels
- Prerequisite: a similarity measure among the edges, i.e. a "Graph of Edges" (not directly provided)
- Can be induced from G and H via Graph Product!

The Proposed Framework

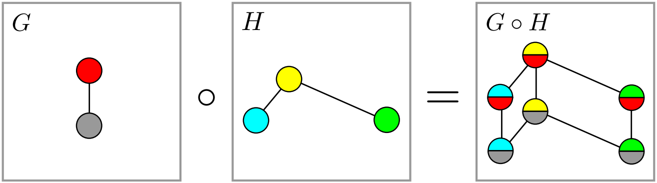
The "Graph of Edges" can be induced by taking the product of G and H



- In the product graph $G \circ H$
  - Each Vertex $\sim$ edge (in the original bipartite graph)
  - Each Edge $\sim$ edge-edge similarity

The Proposed Framework

The “Graph of Edges” can be induced by taking the product of G and H



- In the product graph $G \circ H$
  - Each Vertex $\sim$ edge (in the original bipartite graph)
  - Each Edge $\sim$ edge-edge similarity
- The adjacency matrix of the product graph is defined by “$\circ$” (to be discussed later).

The Proposed Framework

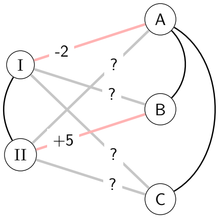
Problem Mapping

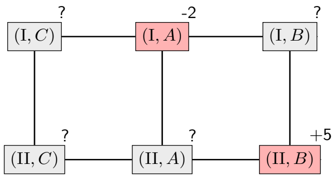
Edge Prediction (Original Problem)

Given G, H and labeled edges, predict the unlabeled edges

Vertex Prediction (Equivalent Problem)

Given $G \circ H$ and labeled vertices, predict the unlabeled vertices





Outline

Outline

Product Graph Construction

Q: When should vertex $(i, j) \sim (i', j')$ in the product graph?

Tensor GP $i \sim i'$ in G AND $j \sim j'$ in H

Cartesian GP $(i \sim i'$ in G AND $j = j')$ OR $(i = i'$ AND $j \sim j'$ in $H)$

Product Graph Construction

Q: When should vertex $(i,j) \sim (i',j')$ in the product graph?

Tensor GP $i \sim i'$ in G AND $j \sim j'$ in H

Cartesian GP $\left(i \sim i' \text{ in } G \text{ AND } j = j'\right)$ OR $\left(i = i' \text{ AND } j \sim j' \text{ in } H\right)$

Can be trivially generalized to weighted graphs.

Product Graph Construction

Q: When should vertex $(i,j) \sim (i',j')$ in the product graph?

Tensor GP $i \sim i'$ in G AND $j \sim j'$ in H

Cartesian GP $\left(i \sim i' \text{ in } G \text{ AND } j = j'\right)$ OR $\left(i = i' \text{ AND } j \sim j' \text{ in } H\right)$

Can be trivially generalized to weighted graphs.

To compute the adjacency matrices of PG

- $G \circ_{Tensor} H = \underbrace{G \otimes H}_{\text{Kronecker (a.k.a. Tensor) Product}}$
- $G \circ_{Cartesian} H = G \otimes I + I \otimes H = \underbrace{G \oplus H}_{\text{Kronecker Sum}}$

Product Graph Construction

Both GPs can be written in the form of spectral decomposition

$$G \circ_{Tensor} H = \sum_{i,j} (\lambda_i \times \mu_j)(u_i \otimes v_j)(u_i \otimes v_j)^\top \tag{1}$$

$$G \circ_{Cartesian} H = \sum_{i,j} (\lambda_i + \mu_j)(u_i \otimes v_j)(u_i \otimes v_j)^\top \tag{2}$$

Product Graph Construction

Both GPs can be written in the form of spectral decomposition

$$G \circ_{Tensor} H = \sum_{i,j} \underbrace{(\lambda_i \times \mu_j)}_{\text{soft AND}} (u_i \otimes v_j)(u_i \otimes v_j)^\top \quad (1)$$

$$G \circ_{Cartesian} H = \sum_{i,j} \underbrace{(\lambda_i + \mu_j)}_{\text{soft OR}} (u_i \otimes v_j)(u_i \otimes v_j)^\top \quad (2)$$

The interplay of graphs is captured by the interplay of their spectrum!

Product Graph Construction

Both GPs can be written in the form of spectral decomposition

$$G \circ_{Tensor} H = \sum_{i,j} \underbrace{(\lambda_i \times \mu_j)}_{\text{soft AND}} (u_i \otimes v_j)(u_i \otimes v_j)^\top \quad (1)$$

$$G \circ_{Cartesian} H = \sum_{i,j} \underbrace{(\lambda_i + \mu_j)}_{\text{soft OR}} (u_i \otimes v_j)(u_i \otimes v_j)^\top \quad (2)$$

The interplay of graphs is captured by the interplay of their spectrum!

Generalization: Spectral Graph Product

$$G \circ H \stackrel{def}{=} \sum_{i,j} (\lambda_i \circ \mu_j)(u_i \otimes v_j)(u_i \otimes v_j)^\top \quad (3)$$

where "$\circ$" can be arbitrary binary operator ("$\times$", "$+$", ...)

Product Graph Construction

Both GPs can be written in the form of spectral decomposition

$$G \circ_{Tensor} H = \sum_{i,j} \underbrace{(\lambda_i \times \mu_j)}_{\text{soft AND}} (u_i \otimes v_j)(u_i \otimes v_j)^\top \quad (1)$$

$$G \circ_{Cartesian} H = \sum_{i,j} \underbrace{(\lambda_i + \mu_j)}_{\text{soft OR}} (u_i \otimes v_j)(u_i \otimes v_j)^\top \quad (2)$$

The interplay of graphs is captured by the interplay of their spectrum!

Generalization: Spectral Graph Product

$$G \circ H \stackrel{def}{=} \sum_{i,j} (\lambda_i \circ \mu_j)(u_i \otimes v_j)(u_i \otimes v_j)^\top \quad (3)$$

where "$\circ$" can be arbitrary binary operator ("$\times$", "$+$", ...)

Commutative Property: $G \circ H$ and $H \circ G$ are isomorphic.

Outline

Graph-based Transductive Learning

With the product graph $A \stackrel{def}{=} G \circ H$ constructed, we solve a standard graph-based transductive learning problem over A

Graph-based Transductive Learning

With the product graph $A \stackrel{def}{=} G \circ H$ constructed, we solve a standard graph-based transductive learning problem over A

Learning Objective

$$\min_f \quad \underbrace{\ell(f)}_{\text{Loss Function}} + \underbrace{\lambda f^\top A^{-1} f}_{\text{Graph Regularization}} \tag{4}$$

f_i system-predicted value for vertex i in A

$\ell(f)$ quantifies the gap between f and partially observed labels.

$\lambda f^\top A^{-1} f$ quantifies the smoothness over graph

- Underlying assumption: $f \sim \mathcal{N}(0, A)$

Graph-based Transductive Learning

The enhanced learning objective

$$\min_{f} \quad \underbrace{\ell(f)}_{\text{Loss Function}} + \underbrace{\lambda f^\top \kappa(A)^{-1} f}_{\text{Graph Regularization}} \tag{5}$$

to incorporate a variety of graph transduction patterns:

Graph-based Transductive Learning

The enhanced learning objective

$$\min_f \underbrace{\ell(f)}_{\text{Loss Function}} + \underbrace{\lambda f^\top \kappa(A)^{-1} f}_{\text{Graph Regularization}} \tag{5}$$

to incorporate a variety of graph transduction patterns:

- k-step Random Walk $\kappa(A) = A^k$
- Regularized Laplacian $\kappa(A) = (\epsilon I - A)^{-1} = I + A + A^2 + A^3 + \ldots$
- Diffusion Process $\kappa(A) = \exp(A) \equiv I + A + \frac{1}{2!}A^2 + \frac{1}{3!}A^3 + \cdots$

Graph-based Transductive Learning

The enhanced learning objective

$$\min_f \quad \underbrace{\ell(f)}_{\text{Loss Function}} + \underbrace{\lambda f^\top \kappa(A)^{-1} f}_{\text{Graph Regularization}} \tag{5}$$

to incorporate a variety of graph transduction patterns:

- k-step Random Walk $\kappa(A) = A^k$
- Regularized Laplacian $\kappa(A) = (\epsilon I - A)^{-1} = I + A + A^2 + A^3 + \dots$
- Diffusion Process $\kappa(A) = \exp(A) \equiv I + A + \frac{1}{2!}A^2 + \frac{1}{3!}A^3 + \cdots$

All can be viewed as to transform the spectrum of $A := \sum_i \theta_i u_i u_i^\top$

$$A^k = \sum_i \theta_i^k u_i u_i^\top \quad (\epsilon I - A)^{-1} = \sum_i \frac{1}{\epsilon - \theta_i} u_i u_i^\top \quad \exp(A) = \sum_i e^{\theta_i} u_i u_i^\top$$

Outline

Optimization

Transductive Learning over Product Graph

$$\min_{f} \quad \ell(f) + \lambda \underbrace{f^\top \kappa(A)^{-1} f}_{r(f)} \tag{6}$$

Optimization

Transductive Learning over Product Graph

$$\min_f \quad \ell(f) + \lambda \underbrace{f^\top \kappa(A)^{-1} f}_{r(f)} \tag{6}$$

Challenge: $\kappa(A) = \kappa(\underbrace{G}_{m \times m} \circ \underbrace{H}_{n \times n})$ is a huge $mn \times mn$ matrix!

Optimization

Transductive Learning over Product Graph

$$\min_{f} \quad \ell(f) + \lambda \underbrace{f^{\top} \kappa(A)^{-1} f}_{r(f)} \tag{6}$$

Challenge: $\kappa(A) = \kappa(\underbrace{G}_{m \times m} \circ \underbrace{H}_{n \times n})$ is a huge $mn \times mn$ matrix!

- Prohibitive to load it into memory
- Prohibitive to compute its inverse
- Even if $\kappa(A)^{-1}$ is given, it is expensive to compute $\nabla r(f)$ naively

Optimization

Transductive Learning over Product Graph

$$\min_{f} \quad \ell(f) + \lambda \underbrace{f^\top \kappa(A)^{-1} f}_{r(f)} \tag{6}$$

Challenge: $\kappa(A) = \kappa(\underbrace{G}_{m\times m} \circ \underbrace{H}_{n\times n})$ is a huge $mn \times mn$ matrix!

- ~~Prohibitive to load it into memory~~ No need to store $\kappa(A)$
- Prohibitive to compute its inverse
- Even if $\kappa(A)^{-1}$ is given, it is expensive to compute $\nabla r(f)$ naively

Optimization

Transductive Learning over Product Graph

$$\min_{f} \quad \ell(f) + \lambda \underbrace{f^\top \kappa(A)^{-1} f}_{r(f)} \tag{6}$$

Challenge: $\kappa(A) = \kappa(\underbrace{G}_{m \times m} \circ \underbrace{H}_{n \times n})$ is a huge $mn \times mn$ matrix!

- ~~Prohibitive to load it into memory~~ No need to store $\kappa(A)$
- ~~Prohibitive to compute its inverse~~ No need of matrix inverse
- Even if $\kappa(A)^{-1}$ is given, it is expensive to compute $\nabla r(f)$ naively

Optimization

Transductive Learning over Product Graph

$$\min_{f} \quad \ell(f) + \lambda \underbrace{f^\top \kappa(A)^{-1} f}_{r(f)} \tag{6}$$

Challenge: $\kappa(A) = \kappa(\underbrace{G}_{m\times m} \circ \underbrace{H}_{n\times n})$ is a huge $mn \times mn$ matrix!

- ~~Prohibitive to load it into memory~~ No need to store $\kappa(A)$
- ~~Prohibitive to compute its inverse~~ No need of matrix inverse
- ~~Even if $\kappa(A)^{-1}$ is given, it is expensive to compute $\nabla r(f)$ naively~~
  Can be performed much more efficiently

Optimization

Keys for complexity reduction

1. Instead of matrices—
   - κ only manipulates eigenvalues
   - $\circ$ only manipulates the interplay of eigenvalues

Optimization

Keys for complexity reduction

1. Instead of matrices—
    - κ only manipulates eigenvalues
    - $\circ$ only manipulates the interplay of eigenvalues
2. The "vec" trick:
    - Bottleneck: multiplication $(X \otimes Y)f$

Optimization

Keys for complexity reduction

1. Instead of matrices—
   - κ only manipulates eigenvalues
   - $\circ$ only manipulates the interplay of eigenvalues
2. The "vec" trick:
   - Bottleneck: multiplication $(X \otimes Y)f$
   - $f = vec(F)$, where $F_{ij} \overset{def}{=}$ system-predicted score for edge (i, j)

Optimization

Keys for complexity reduction

1. Instead of matrices—
   - κ only manipulates eigenvalues
   - $\circ$ only manipulates the interplay of eigenvalues
2. The "vec" trick:
   - Bottleneck: multiplication $(X \otimes Y)f$
   - $f = vec(F)$, where $F_{ij} \stackrel{def}{=}$ system-predicted score for edge (i, j)

$$\underbrace{(X \otimes Y)f}_{O(m^2n^2)\text{ time/space}} = (X \otimes Y)vec(F) \equiv \underbrace{vec(XFY^\top)}_{O(mn(m+n))\text{ time, }O((m+n)^2)\text{ space}} \tag{7}$$

Optimization with Low-rank Constraint

Further speedup is possible by factorizing F into two low-rank matrices

Optimization with Low-rank Constraint

Further speedup is possible by factorizing F into two low-rank matrices

- The cost of each alternating gradient step is proportional to $rank(F) \cdot rank(\Sigma)$

Optimization with Low-rank Constraint

Further speedup is possible by factorizing F into two low-rank matrices

- The cost of each alternating gradient step is proportional to $rank(F) \cdot rank(\Sigma)$
- Σ: a "Characteristic Matrix" where $\Sigma_{ij} = \frac{1}{\kappa(\lambda_i \circ \mu_j)}$

Optimization with Low-rank Constraint

Further speedup is possible by factorizing F into two low-rank matrices

- The cost of each alternating gradient step is proportional to $rank(F) \cdot rank(\Sigma)$
- Σ: a "Characteristic Matrix" where $\Sigma_{ij} = \frac{1}{\kappa(\lambda_i \circ \mu_j)}$
  - An interesting observation: $rank(\Sigma)$ is usually a small constant!

Optimization with Low-rank Constraint

Further speedup is possible by factorizing F into two low-rank matrices

- The cost of each alternating gradient step is proportional to $rank(F) \cdot rank(\Sigma)$
- Σ: a "Characteristic Matrix" where $\Sigma_{ij} = \frac{1}{\kappa(\lambda_i \circ \mu_j)}$
  - An interesting observation: $rank(\Sigma)$ is usually a small constant!
  - Example: Diffusion process over the Cartesian PG

$$\Sigma = \begin{bmatrix} e^{-(\lambda_1+\mu_1)} & \dots & e^{-(\lambda_1+\mu_n)} \\ \vdots & \ddots & \vdots \\ e^{-(\lambda_m+\mu_1)} & \dots & e^{-(\lambda_m+\mu_n)} \end{bmatrix} = \begin{bmatrix} e^{-\lambda_1} \\ \vdots \\ e^{-\lambda_m} \end{bmatrix} \begin{bmatrix} e^{-\mu_1} & \dots & e^{-\mu_n} \end{bmatrix}$$

$\implies rank(\Sigma) = 1$

Outline

Datasets and Baselines

Datasets

Dataset	G	H
Movielens-100K	Users	Movies
Cora	Publications	Publications
Courses	Courses	Prerequisite Courses

Baselines

MC Matrix Completion.
- Ignores the info of G and H.

TK Tensor Kernel.
- Implicitly construct PG, no transduction

GRMC Graph Regularized Matrix Completion.
- Transduction over G and H, no PG constructed

Results

Performance of several interesting combinations of $\circ$ and κ

Dataset	Graph Transduction	Graph Product	MAP	AUC	ndcg@3
Courses	Random Walk	Tensor	0.488	0.827	0.461
	Diffusion	Cartesian	**0.518**	**0.872**	**0.500**
	von-Neumann	Tensor	0.472	0.861	0.449
	von-Neumann	Cartesian	0.366	0.531	0.359
	Sigmoid	Cartesian	0.443	0.617	0.431
Cora	Random Walk	Tensor	0.222	0.764	0.205
	Diffusion	Cartesian	**0.256**	**0.884**	**0.232**
	von-Neumann	Tensor	0.230	0.853	0.211
	von-Neumann	Cartesian	0.218	0.633	0.212
	Sigmoid	Cartesian	0.192	0.443	0.188
MovieLens	Random Walk	Tensor	-	-	0.7695
	Diffusion	Cartesian	-	-	0.7702
	von-Neumann	Tensor	-	-	**0.7720**
	von-Neumann	Cartesian	-	-	0.7624
	Sigmoid	Cartesian	-	-	0.7650

Results

Proposed method (Diff + Cartesian GP) v.s. Baselines

Dataset	Method	MAP	AUC	ndcg@3
Courses	MC	0.319	0.758	0.294
	GRMC	0.366	0.777	0.343
	TK	0.449	0.810	0.446
	Proposed	**0.490**	**0.838**	**0.473**
Cora	MC	0.101	0.697	0.086
	GRMC	0.115	0.702	0.101
	TK	0.248	0.872	0.231
	Proposed	**0.268**	**0.894**	**0.243**
MovieLens	MC	-	-	0.748
	GRMC	-	-	0.752
	TK	-	-	0.718
	Proposed	-	-	**0.765**

Outline

Conclusion

Summary

Problem Predicting the missing edges of a bipartite graph with graph-structured vertex sets on both sides.

Contribution A novel approach via transductive learning over product graph, efficient algorithmic solution and good results.

Conclusion

Summary

Problem Predicting the missing edges of a bipartite graph with graph-structured vertex sets on both sides.

Contribution A novel approach via transductive learning over product graph, efficient algorithmic solution and good results.

On-going Work

- Extend to k Graphs $(k > 2)$
  - Bipartite Graph $\rightarrow$ k-partite Graph
  - Edge $\rightarrow$ Hyperedge
- Determine the “optimal” graph product for any given problem.

Thanks!

hanxiaol@cs.cmu.edu