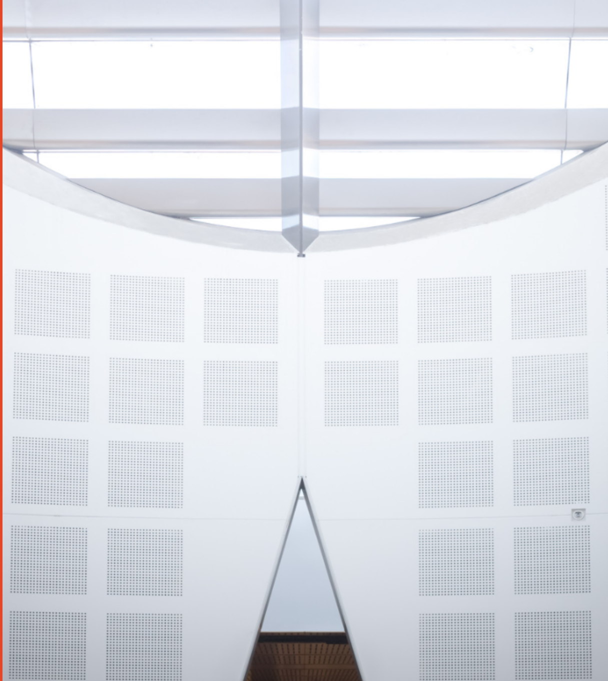
COMP5313/COMP4313 - Large Scale Networks

Week 9: Machine Learning on Graphs (II): Propagating Labels and Features

Lijun Chang

May 1, 2025

THE UNIVERSITY OF SYDNEY



Introduction

- Last week, we talked about graph neural networks for graph machine learning
- Today, we will look at label propagation and feature propagation
  - Label propagation can be used when no node features are available
  - Label propagation can be used as a post-process to improve the prediction accuracy
  - Label propagation can be integrated into graph neural networks
  - Feature propagation can be used as a pre-process to improve the scalability

Outline

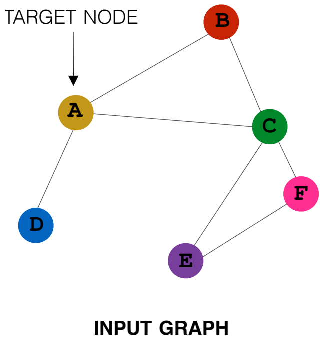
Setup for Node Classification

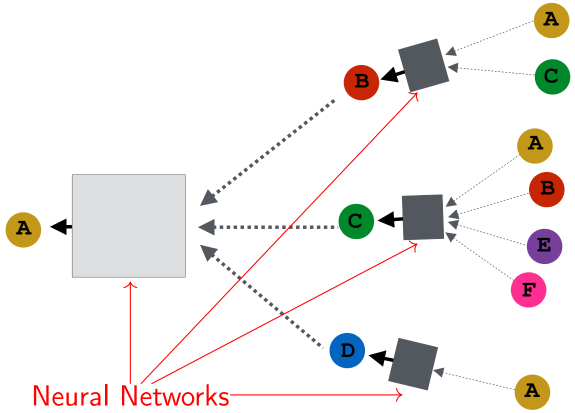
Assume we have an undirected graph G:

- V is the node set ($|V| = N$)
- $\mathbf{A} \in \mathbb{R}^{N \times N}$ is the adjacency matrix
- $\mathbf{X} \in \mathbb{R}^{N \times q}$ is a matrix of input node features
  - Categorical attributes, text, image data
  - Node degrees, clustering coefficients, etc.
  - Indicator vectors (i.e., one-hot encoding of each node)
- For each labeled node $v \in V_{\text{train}}$, its label is denoted by $\mathbf{y}_v \in \mathbb{R}^C$, a one-hot encoding vector of the label
- The task is to predict the labels for nodes in $V \setminus V_{\text{train}}$

Neighborhood Aggregation

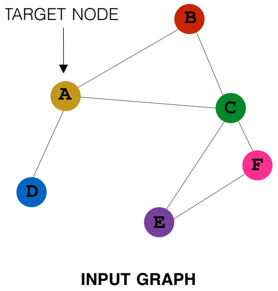
- Graph neural networks utilize both graph structure and node features via neighborhood aggregation.
- Intuition: nodes aggregate information from their neighbors using neural networks

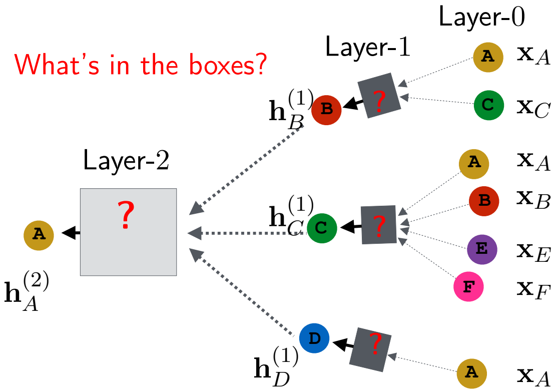




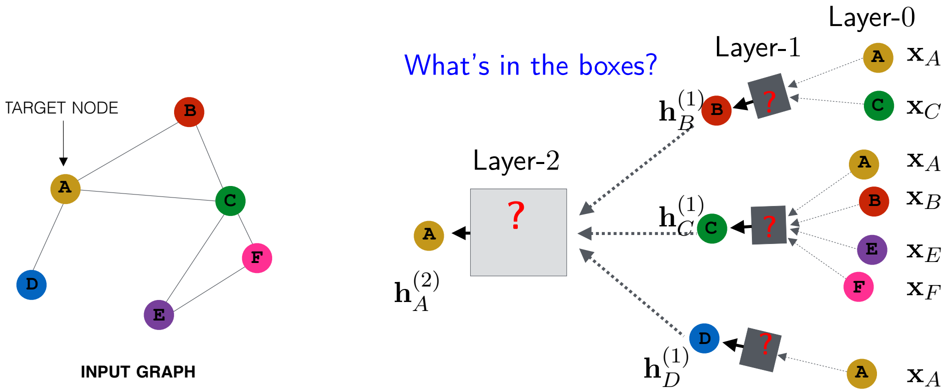
Neighborhood Aggregation

- Key distinctions of different graph neural network models are in how they aggregate information across the layers





Graph Neural Networks (GNNs)

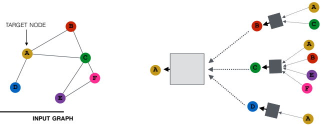


- In general, each box conducts two operations
  - $\mathbf{a}_u^{(k)} = \mathrm{AGGREGATE}^{(k)}\left(\left\{\mathbf{h}_v^{(k-1)} \mid v \in N(u)\right\}\right)$
  - $\mathbf{h}_u^{(k)} = \mathrm{COMBINE}^{(k)}\left(\mathbf{h}_u^{(k-1)}, \mathbf{a}_u^{(k)}\right)$

Graph Convolutional Networks (GCN) [1]

Weighted average of neighbor messages and then applying a neural network

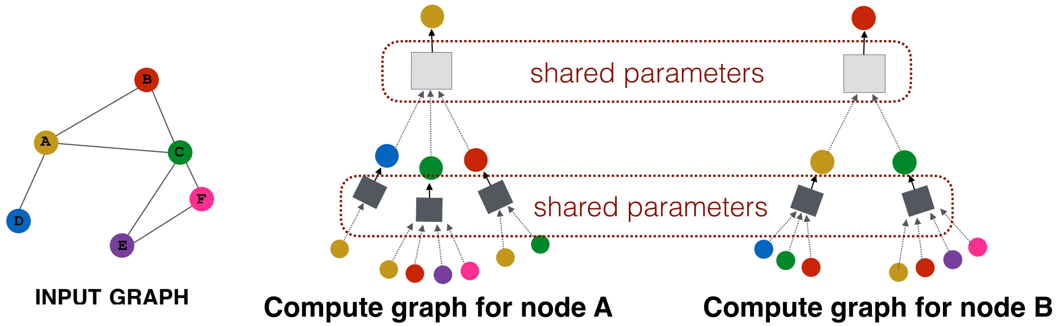
- $\mathbf{h}_u^{(0)} = \mathbf{x}_u, \forall u \in V$
- For $k = 1, \ldots, K$
  - $\mathbf{h}_u^{(k)} = \sigma\left(\mathbf{W}^{(k)}\left(\sum_{v \in N(u) \cup \{u\}} \frac{\mathbf{h}_v^{(k-1)}}{\sqrt{(d(v)+1)(d(u)+1)}}\right)\right), \quad \forall u \in V$
    - $N(u)$ is the neighbors of u, $d(u) = |N(u)|$ is the degree of u
    - $\mathbf{W}^{(k)}$ is the trainable weight parameters
    - $\sigma(\cdot)$ is element-wise activation function (e.g., $\mathrm{ReLu}(x) = \max(0, x)$)
- $\mathbf{z}_u = \mathbf{h}_u^{(K)}$ is the learned representation of u, $\forall u \in V$.
  - These representation can be fed into any loss function to train the weight parameters. E.g., $\{(\mathbf{z}_u, \mathbf{y}_u)\}$ can be fed into logistic regression for classification.



[1] Thomas N. Kipf, Max Welling: Semi-Supervised Classification with Graph Convolutional Networks. ICLR (Poster) 2017

Inductive Capability

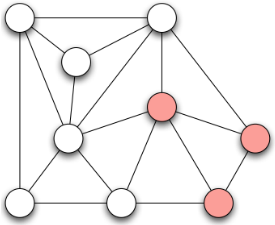
- The same weight parameters are shared for the same layer
- The number of weight parameters is sublinear in N
- This makes training on large graphs possible.



Outline

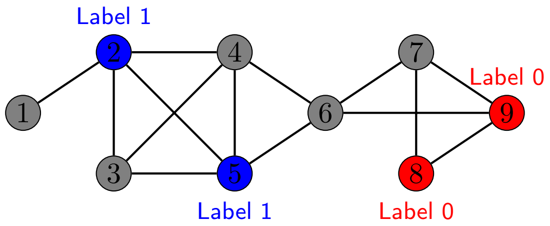
The Homophily Assumption

- Connected nodes tend to share the same label



Semi-supervised Learning

How can we leverage homophily observed in networks to help predict node labels?



- How do we predict the labels for the nodes in grey without using node features?
  - $y_v = 1$ if node v belongs to class 1.
  - $y_v = 0$ if node v belongs to class 0.

Based on http://cs224w.stanford.edu

Label Propagation

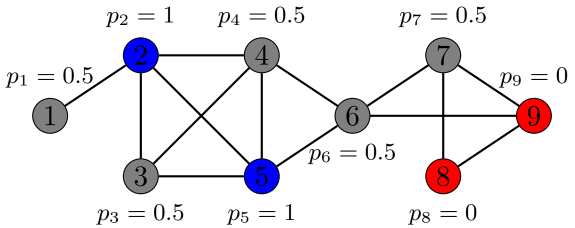
- Instead of directly predicting y_v, we compute $\mathbf{Pr}(y_v = 1)$
  - Predict label 1 if $\mathbf{Pr}(y_v = 1) > 0.5$
  - Predict label 0 otherwise
- Idea of propagating node labels across the network
  - Class probability $\mathbf{Pr}(y_v = 1)$ is a weighted average of the class probabilities of v's neighbors
  - $\mathbf{Pr}^{(k)}(y_v = 1) = \frac{1}{\sum_u A_{v,u}} \sum_u A_{v,u} \cdot \mathbf{Pr}^{(k-1)}(y_u = 1)$.
  - Initialize $\mathbf{Pr}^{(0)}(y_v = 1)$ for labeled nodes based on their ground-truth label y_v^*, and initialize $\mathbf{Pr}^{(0)}(y_v = 1) = 0.5$ for unlabeled nodes.
  - Update all unlabeled nodes (in parallel or a random order) until convergence or a maximum number of iterations.

Based on http://cs224w.stanford.edu

Label Propagation

Initialization

- All labeled nodes are initialized based on their labels
- All unlabeled nodes are initialized as 0.5 (belong to class 1 with probability 0.5)

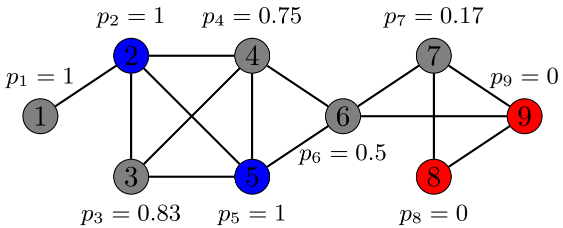


Notation: $p_i = \mathbf{Pr}(y_i = 1)$ for $1 \le i \le 9$.

Based on http://cs224w.stanford.edu

Label Propagation

After Iteration 1

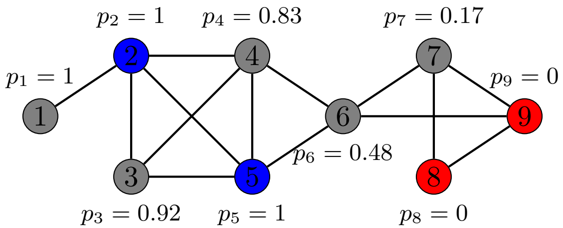


Notation: $p_i = \mathbf{Pr}(y_i = 1)$ for $1 \leq i \leq 9$.

Based on http://cs224w.stanford.edu

Label Propagation

After Iteration 2

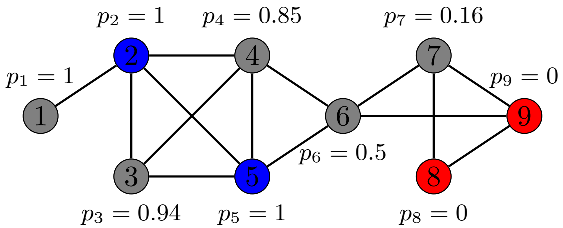


Notation: $p_i = \mathbf{Pr}(y_i = 1)$ for $1 \leq i \leq 9$.

Based on http://cs224w.stanford.edu

Label Propagation

After Iteration 3

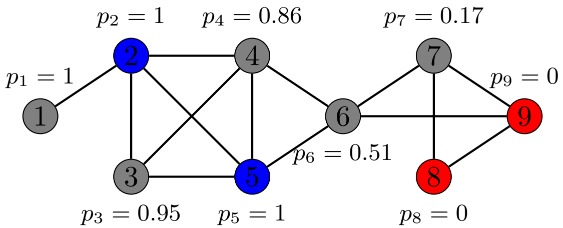


Notation: $p_i = \mathbf{Pr}(y_i = 1)$ for $1 \leq i \leq 9$.

Based on http://cs224w.stanford.edu

Label Propagation

After Iteration 4

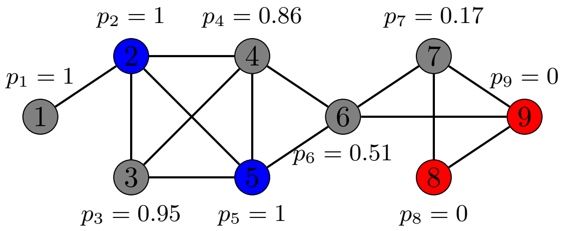


Notation: $p_i = \mathbf{Pr}(y_i = 1)$ for $1 \leq i \leq 9$.

Based on http://cs224w.stanford.edu

Label Propagation

- All scores stabilize after 4 iterations.
  - $|\mathbf{Pr}^{(k)}(y_v) - \mathbf{Pr}^{(k-1)}(y_v)| \leq \varepsilon$ for all nodes v.
- Thus, we predict
  - Nodes 1, 3, 4, 6 belong to class 1
  - Node 7 belong to class 0



Based on http://cs224w.stanford.edu

Theoretical Foundation of Label Propagation

- Let $\mathbf{Y} \in \mathbb{R}^{N \times C}$ be the one-hot-encoding matrix of node labels, where $Y_{i,j} = 1$ if node i is labeled as j.
- Let's consider the optimization problem

$$\hat{\mathbf{Y}} = \underset{\mathbf{T} \in \mathbb{R}^{N \times C}}{\arg\min}\, \alpha \cdot \mathrm{Tr}(\mathbf{T}^\top \mathbf{L}^{\mathrm{sym}} \mathbf{T}) + (1 - \alpha) \cdot \|\mathbf{T} - \mathbf{Y}\|_F^2$$

where $\mathbf{L}^{\mathrm{sym}} = \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$ is the symmetric normalized Laplacian matrix. This is equivalent to

$$\begin{aligned} \hat{\mathbf{Y}} &= \underset{\mathbf{T} \in \mathbb{R}^{N \times C}}{\arg\min}\, \alpha \cdot \textstyle\sum_{j=1}^{C} \mathbf{t}_j^\top \mathbf{L}^{\mathrm{sym}} \mathbf{t}_j + (1 - \alpha) \cdot \sum_{i=1}^{N} \sum_{j=1}^{C} (T_{i,j} - Y_{i,j})^2 \\ &= \underset{\mathbf{T} \in \mathbb{R}^{N \times C}}{\arg\min}\, \alpha \cdot \textstyle\sum_{j=1}^{C} \sum_{(i,k) \in E} \left(\frac{T_{i,j}}{\sqrt{d_i}} - \frac{T_{k,j}}{\sqrt{d_k}}\right)^2 + (1 - \alpha) \cdot \sum_{i=1}^{N} \sum_{j=1}^{C} (T_{i,j} - Y_{i,j})^2 \end{aligned}$$

where $\mathbf{t}_j$ denotes the j-th column of $\mathbf{T}$.

Dengyong Zhou, Olivier Bousquet, Thomas Navin Lal, Jason Weston, Bernhard Schölkopf: Learning with Local and Global Consistency. NIPS 2003: 321-328

Theoretical Foundation of Label Propagation

$$\begin{aligned}\hat{\mathbf{Y}} &= \underset{\mathbf{T}\in\mathbb{R}^{N\times C}}{\arg\min}\ \alpha\cdot\mathrm{Tr}(\mathbf{T}^\top\mathbf{L}^{\mathrm{sym}}\mathbf{T}) + (1-\alpha)\cdot\|\mathbf{T}-\mathbf{Y}\|_F^2 \\ &= \underset{\mathbf{T}\in\mathbb{R}^{N\times C}}{\arg\min}\ \alpha\cdot\textstyle\sum_{j=1}^{C}\sum_{(i,k)\in E}\left(\frac{T_{i,j}}{\sqrt{d_i}}-\frac{T_{k,j}}{\sqrt{d_k}}\right)^2 + (1-\alpha)\cdot\sum_{i=1}^{N}\sum_{j=1}^{C}(T_{i,j}-Y_{i,j})^2\end{aligned}$$

- The first term of the RHS is the smoothness constraint, which means that a good classifying function should not change too much between connected nodes.
- The second term is the fitting constraint, which means a good classifying function should not change too much from the initial label assignment.
- The gradient of $\alpha\cdot\mathrm{Tr}(\mathbf{T}^\top\mathbf{L}^{\mathrm{sym}}\mathbf{T}) + (1-\alpha)\cdot\|\mathbf{T}-\mathbf{Y}\|_F^2$ with respect to $\mathbf{T}$ is $2\alpha\mathbf{L}^{\mathrm{sym}}\mathbf{T} + 2(1-\alpha)(\mathbf{T}-\mathbf{Y})$.
- Thus, the optimal solution is $\hat{\mathbf{Y}} = (1-\alpha)(\mathbf{I}-\alpha\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2})^{-1}\mathbf{Y}$

Dengyong Zhou, Olivier Bousquet, Thomas Navin Lal, Jason Weston, Bernhard Schölkopf: Learning with Local and Global Consistency. NIPS 2003: 321-328

Recap of Personalized PageRank

- Personalized PageRank is used to illuminate a region of a large graph around a target set S of interest.
  - At each time step, the random surfer has two options
    - With probability α, follow a link at random
    - With probability $1-\alpha$, jump to a random page among a set S of pre-selected pages

- Iterative computation: $\mathbf{r}^{(k)} = \alpha \mathbf{A}^\top \mathbf{D}^{-1} \mathbf{r}^{(k-1)} + (1-\alpha)\mathbf{s}$
  - s is preference/personalization vector of a user
  - E.g., to compute the relevance/importance of all nodes to nodes 1 and 2, we can use $\mathbf{s} = (\frac{1}{2}, \frac{1}{2}, 0 \dots, 0)$

- Equilibrium: $\mathbf{r}^* = (1-\alpha)(\mathbf{I} - \alpha \mathbf{A}^\top \mathbf{D}^{-1})^{-1}\mathbf{s} = (1-\alpha)\sum_{\ell=0}^{\infty} \alpha^\ell (\mathbf{A}^\top \mathbf{D}^{-1})^\ell \mathbf{s}$.

- $\mathbf{r}^{(0)}$ is typically also set as s, and then

$$\mathbf{r}^{(k)} = \alpha^k (\mathbf{A}^\top \mathbf{D}^{-1})^k \mathbf{s} + (1-\alpha)\sum_{\ell=0}^{k-1} \alpha^\ell (\mathbf{A}^\top \mathbf{D}^{-1})^\ell \mathbf{s}$$

Theoretical Foundation of Label Propagation

$$
\begin{aligned}
\hat{\mathbf{Y}} &= \underset{\mathbf{T}\in\mathbb{R}^{N\times C}}{\arg\min}\ \alpha \cdot \mathrm{Tr}(\mathbf{T}^\top \mathbf{L}^{\mathrm{sym}} \mathbf{T}) + (1-\alpha)\cdot \|\mathbf{T}-\mathbf{Y}\|_F^2 \\
&= \underset{\mathbf{T}\in\mathbb{R}^{N\times C}}{\arg\min}\ \alpha \cdot \sum_{j=1}^{C} \sum_{(i,k)\in E} \left(\frac{T_{i,j}}{\sqrt{d_i}} - \frac{T_{k,j}}{\sqrt{d_k}}\right)^2 + (1-\alpha)\cdot \sum_{i=1}^{N} \sum_{j=1}^{C} (T_{i,j} - Y_{i,j})^2
\end{aligned}
$$

- The optimal solution is
  $\hat{\mathbf{Y}} = (1-\alpha)(\mathbf{I}-\alpha\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2})^{-1}\mathbf{Y} = (1-\alpha)\sum_{\ell=0}^{\infty}\alpha^\ell(\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2})^\ell\mathbf{Y}$.

- It can be computed iteratively as $\mathbf{Y}^{(k)} = \alpha\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}\mathbf{Y}^{(k-1)} + (1-\alpha)\mathbf{Y}$.

Dengyong Zhou, Olivier Bousquet, Thomas Navin Lal, Jason Weston, Bernhard Schölkopf: Learning with Local and Global Consistency. NIPS 2003: 321-328

Outline

Recap of the Naïve Approach

- Ignore the graph structure information (i.e., ignore $\mathbf{A}$), and use only $\mathbf{X}$ and $\mathbf{y}_v$ for training and prediction
- This may perform well on some datasets where $\mathbf{X}$ contains a lot of information
- In general, if we incorporate graph structure information, we can do better

Correct&Smooth Makes the Naïve Approach Great Again

- Correct&Smooth follows the three-step procedure[2]

Step 1. Train a base predictor, e.g., a linear model, an MLP, or a GNN

Step 2. Use the base predictor to predict soft labels (class probabilities) of all nodes.

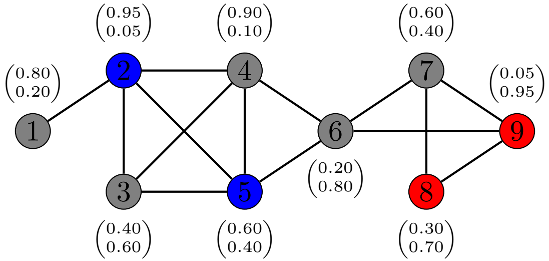
Step 3. Post-process the predictions using graph structure to obtain the final predictions of all nodes.

[2] Qian Huang, Horace He, Abhay Singh, Ser-Nam Lim, Austin R. Benson: Combining Label Propagation and Simple Models out-performs Graph Neural Networks. ICLR 2021

Based on http://cs224w.stanford.edu

Correct&Smooth: Step 2

- Given a trained base predictor, we apply it to obtain soft labels for all the nodes.



$\begin{pmatrix}0.80\\0.20\end{pmatrix}$
$\begin{pmatrix}0.95\\0.05\end{pmatrix}$
$\begin{pmatrix}0.90\\0.10\end{pmatrix}$
$\begin{pmatrix}0.60\\0.40\end{pmatrix}$
$\begin{pmatrix}0.05\\0.95\end{pmatrix}$
$\begin{pmatrix}0.20\\0.80\end{pmatrix}$
$\begin{pmatrix}0.40\\0.60\end{pmatrix}$
$\begin{pmatrix}0.60\\0.40\end{pmatrix}$
$\begin{pmatrix}0.30\\0.70\end{pmatrix}$
1
2
3
4
5
6
7
8
9

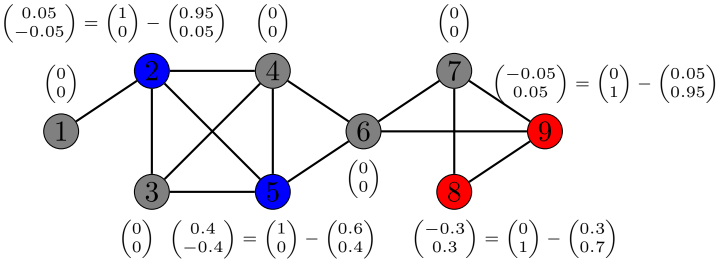
Figure: Soft labels $\mathbf{Z}$ by base predictor

Based on http://cs224w.stanford.edu

Correct&Smooth: Step 3.1

The Correcting Step

- Compute training errors of nodes
  - Training error: ground-truth label minus soft label. Defined as 0 for unlabeled nodes.



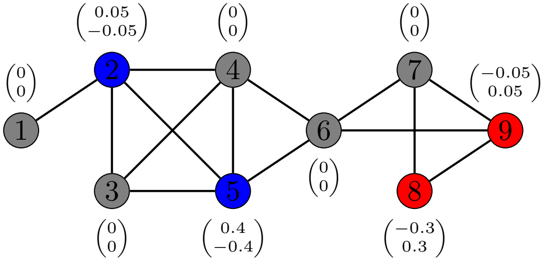
$\begin{pmatrix}0.05\\-0.05\end{pmatrix}=\begin{pmatrix}1\\0\end{pmatrix}-\begin{pmatrix}0.95\\0.05\end{pmatrix}$
$\begin{pmatrix}0\\0\end{pmatrix}$
$\begin{pmatrix}0\\0\end{pmatrix}$
$\begin{pmatrix}0\\0\end{pmatrix}$
$\begin{pmatrix}-0.05\\0.05\end{pmatrix}=\begin{pmatrix}0\\1\end{pmatrix}-\begin{pmatrix}0.05\\0.95\end{pmatrix}$
1
2
4
7
6
9
3
5
8
$\begin{pmatrix}0\\0\end{pmatrix}$
$\begin{pmatrix}0\\0\end{pmatrix}$
$\begin{pmatrix}0.4\\-0.4\end{pmatrix}=\begin{pmatrix}1\\0\end{pmatrix}-\begin{pmatrix}0.6\\0.4\end{pmatrix}$
$\begin{pmatrix}-0.3\\0.3\end{pmatrix}=\begin{pmatrix}0\\1\end{pmatrix}-\begin{pmatrix}0.3\\0.7\end{pmatrix}$

Based on http://cs224w.stanford.edu

Correct&Smooth: Step 3.1

The Correcting Step

- The degree of the errors of the soft labels are biased.
- We need to correct for the error bias.
- The key idea is that we expect errors in the base prediction to be positively correlated along edges in the graph.
    - In other words, an error at node u increases the chance of a similar error at neighbors of u.



Correct&Smooth: Step 3.1

The Correcting Step

- Diffuse the training errors $\mathbf{E}^{(0)}$ along the edges
- $\mathbf{E}^{(k)} = \alpha \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}\mathbf{E}^{(k-1)} + (1-\alpha)\mathbf{E}^{(0)}$

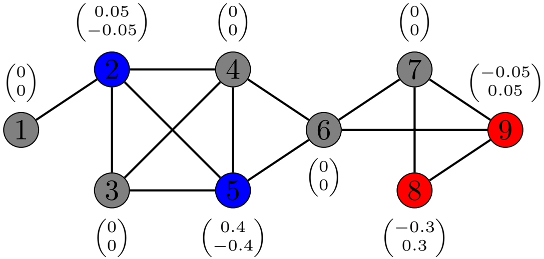


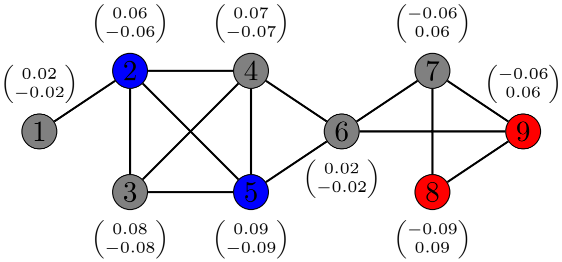
Figure: Initial training error $\mathbf{E}^{(0)}$

Based on http://cs224w.stanford.edu

Correct&Smooth: Step 3.1

The Correcting Step

- Diffuse the training errors $\mathbf{E}^{(0)}$ along the edges
- $\mathbf{E}^{(k)} = \alpha \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}\mathbf{E}^{(k-1)} + (1-\alpha)\mathbf{E}^{(0)}$



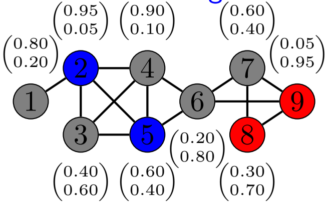
$\begin{pmatrix}0.02\\-0.02\end{pmatrix}$
$\begin{pmatrix}0.06\\-0.06\end{pmatrix}$
$\begin{pmatrix}0.07\\-0.07\end{pmatrix}$
$\begin{pmatrix}-0.06\\0.06\end{pmatrix}$
$\begin{pmatrix}-0.06\\0.06\end{pmatrix}$
1
2
4
7
6
9
3
5
8
$\begin{pmatrix}0.02\\-0.02\end{pmatrix}$
$\begin{pmatrix}0.08\\-0.08\end{pmatrix}$
$\begin{pmatrix}0.09\\-0.09\end{pmatrix}$
$\begin{pmatrix}-0.09\\0.09\end{pmatrix}$

Figure: After diffusion $\mathbf{E}^{(3)}$ $(\alpha = 0.8)$

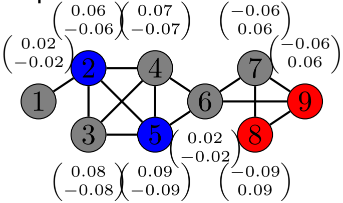
Based on http://cs224w.stanford.edu

Correct&Smooth: Step 3.2

Add the scaled diffused training errors into the predicted soft labels.



(a) Soft labels $\mathbf{Z}$ by base predictor



(b) Diffused training error $\mathbf{E}^{(3)}$

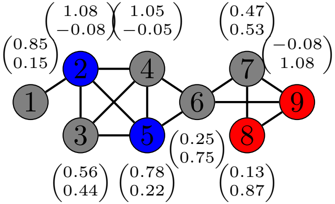
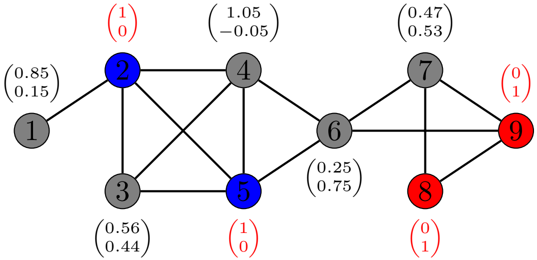


Figure: Corrected soft labels $\mathbf{Z} + 2 \cdot \mathbf{E}^{(3)}$

Correct&Smooth: Step 3.3

The Smoothing Step

- Smoothen the corrected soft labels along the edges
- Assumption: neighboring nodes tend to share the same labels
- For training nodes, we use the ground-truth hard labels instead of the soft labels



$\begin{pmatrix}0.85\\0.15\end{pmatrix}$
$\begin{pmatrix}1\\0\end{pmatrix}$
$\begin{pmatrix}1.05\\-0.05\end{pmatrix}$
$\begin{pmatrix}0.47\\0.53\end{pmatrix}$
$\begin{pmatrix}0\\1\end{pmatrix}$
1
2
3
4
5
6
7
8
9
$\begin{pmatrix}0.25\\0.75\end{pmatrix}$
$\begin{pmatrix}0.56\\0.44\end{pmatrix}$
$\begin{pmatrix}1\\0\end{pmatrix}$
$\begin{pmatrix}0\\1\end{pmatrix}$

Figure: Input $\mathbf{Z}^{(0)}$ to the smoothing step

Based on http://cs224w.stanford.edu

Correct&Smooth: Step 3.3

The Smoothing Step

- Diffuse label $\mathbf{Z}^{(0)}$ along the graph structure
- $\mathbf{Z}^{(k)} = \alpha \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}\mathbf{Z}^{(k-1)} + (1-\alpha)\mathbf{Z}^{(0)}$.

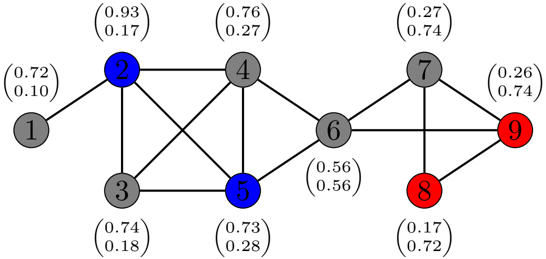


Figure: After diffusion $\mathbf{Z}^{(3)}$ $(\alpha = 0.8)$

Based on http://cs224w.stanford.edu

Outline

Graph Convolutional Networks (GCN)

In matrix form

- $\mathbf{H}^{(0)} = \mathbf{X} \in \mathbb{R}^{N \times q}$
- For $k = 1, \ldots, K$
  - $\mathbf{H}^{(k)} = \sigma\left(\tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2}\mathbf{H}^{(k-1)}\mathbf{W}^{(k)}\right)$
    - $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$
    - $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$
- $\mathbf{Z} = \mathbf{H}^{(K)}$

For $K = 2$, $\mathbf{Z} = \mathrm{softmax}\left(\tilde{\mathbf{A}}^{\mathrm{sym}}\mathrm{ReLu}\left(\tilde{\mathbf{A}}^{\mathrm{sym}}\mathbf{X}\mathbf{W}^{(1)}\right)\mathbf{W}^{(2)}\right)$ where $\tilde{\mathbf{A}}^{\mathrm{sym}} = \tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2}$.

Drawbacks of GCN

- GCN makes use of the K-hop neighbors for each node to make predictions
- When K is small, it only utilizes the information of a limited neighborhood
- A larger neighborhood would be desirable to provide the model with more information
- However, when K is large, there is the oversmoothing problem.

Personalized Propagation of Neural Predictions (PPNP) [3]

- PPNP utilizes a propagation scheme derived from personalized PageRank.
- $\mathbf{Z} = \mathrm{softmax}\left((1-\alpha)(\mathbf{I} - \alpha\tilde{\mathbf{A}}^{\mathrm{sym}})^{-1}\mathbf{H}\right)$ and $\mathbf{h}_i = \mathrm{MLP}_\theta(\mathbf{x}_i)$, where $\mathbf{h}_i$ is the i-th row of $\mathbf{H}$.
- We can also interpret it from the perspective of label propagation.
- Approximate personalized propagation of neural predictions (APPNP)

$$
\begin{aligned}
\mathbf{Z}^{(0)} &= \mathbf{H} = \mathrm{MLP}_\theta(\mathbf{X}) \\
\mathbf{Z}^{(k)} &= \alpha\tilde{\mathbf{A}}^{\mathrm{sym}}\mathbf{Z}^{(k-1)} + (1-\alpha)\mathbf{H} \\
\mathbf{Z}^{(K)} &= \mathrm{softmax}\left(\alpha\tilde{\mathbf{A}}^{\mathrm{sym}}\mathbf{Z}^{(K-1)} + (1-\alpha)\mathbf{H}\right)
\end{aligned}
$$

[3] Johannes Klicpera, Aleksandar Bojchevski, Stephan Günnemann: Predict then Propagate: Graph Neural Networks meet Personalized PageRank. ICLR (Poster) 2019

PPRGo [4]

- APPNP is slow in training, as it needs to process the entire graph for $O(K)$ times in each gradient update.
- PPRGo is proposed to speed up the training.
  - The general idea is to pre-compute and store an approximated version $\mathbf{\Pi}^{\varepsilon}$ of $(1-\alpha)(\mathbf{I}-\alpha\tilde{\mathbf{A}}^{\mathrm{sym}})^{-1}$.
  - The PPRGo model is $\mathbf{Z}=\operatorname{softmax}\left(\mathbf{\Pi}^{\varepsilon}\mathbf{H}\right)$ and $\mathbf{h}_i=\mathrm{MLP}_{\theta}(\mathbf{x}_i)$
- How to approximate $(1-\alpha)(\mathbf{I}-\alpha\tilde{\mathbf{A}}^{\mathrm{sym}})^{-1}$?
  - Recall that $(1-\alpha)(\mathbf{I}-\alpha\tilde{\mathbf{D}}^{-1}\tilde{\mathbf{A}})^{-1}$ is the PPR matrix, and we can use Forward Push to approximate each row of the matrix.
  - $(1-\alpha)(\mathbf{I}-\alpha\tilde{\mathbf{A}}^{\mathrm{sym}})^{-1}=\tilde{\mathbf{D}}^{1/2}\left((1-\alpha)(\mathbf{I}-\alpha\tilde{\mathbf{D}}^{-1}\tilde{\mathbf{A}})^{-1}\right)\tilde{\mathbf{D}}^{-1/2}$

[4] Aleksandar Bojchevski, Johannes Klicpera, Bryan Perozzi, Amol Kapoor, Martin Blais, Benedek Rózemberczki, Michal Lukasik, Stephan Günnemann: Scaling Graph Neural Networks with Approximate PageRank. KDD 2020: 2464-2473

Outline

Simplifying Graph Convolutional Networks (SGC) [5]

- GCN update rule: $\mathbf{H}^{(k)} = \sigma\left(\tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2}\mathbf{H}^{(k-1)}\mathbf{W}^{(k)}\right)$
- Training GNNs is time-consuming
  - The number of parameters in GNNs increases with the number of layers K
  - The number of nodes involved in the training (i.e., number of distinct nodes involved in the computation graph of the training nodes) increases exponentially with K
  - K typically is set as 2 or 3 in practice
- How about removing the non-linearity (i.e., $\sigma(\cdot)$) from the update rule?
  - $\mathbf{H}^{(K)} = \tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2}\mathbf{H}^{(K-1)}\mathbf{W}^{(k)} = \left(\tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2}\right)^{K}\mathbf{H}^{(0)}\mathbf{W}^{(1)}\ldots\mathbf{W}^{(K)}$
  - This is equivalent to $\mathbf{H}^{(K)} = \left(\tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2}\right)^{K}\mathbf{X}\mathbf{W}$.
  - The SGC model: $\mathbf{Z} = \mathrm{softmax}(\mathbf{H}^{(K)})$. The linear transformation $\left(\tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2}\right)^{K}\mathbf{X}$ of the input feature $\mathbf{X}$ can be done in a pre-process step.
  - This simple model actually performs well in many classification tasks in practice.

[5] Felix Wu, Amauri H. Souza Jr., Tianyi Zhang, Christopher Fifty, Tao Yu, Kilian Q. Weinberger: Simplifying Graph Convolutional Networks. ICML 2019: 6861-6871

Approximate Graph Propagation (AGP) [6]

- AGP studies the following unified graph propagation

$$\mathbf{H} = \sum_{\ell=0}^{\infty} \alpha_\ell (\mathbf{D}^{-a}\mathbf{A}\mathbf{D}^{-b})^\ell \mathbf{x}$$

  and proposes efficient algorithm to approximate $\mathbf{H}$.
  - α_ℓ, a, b are hyper-parameters.
  - $\mathbf{x}$ is one column of the input feature matrix $\mathbf{X}$.
- The general idea of the approximation follows that of approximating personalized PageRank.

[6] Hanzhi Wang, Mingguo He, Zhewei Wei, Sibo Wang, Ye Yuan, Xiaoyong Du, Ji-Rong Wen: Approximate Graph Propagation. KDD 2021: 1686-1696

Conclusion

- Propagating labels and features are powerful techniques for large-scale graphs
- These simple models perform well in practice, but their expressive power is limited in theory