Graph-based Machine Learning

Lecture 2. Node Feature Inference

Yuexin Wu

Part 2 Outline

- Node feature learning
  - Factorization based methods
  - Random walk based methods
  - Other sampling-based methods
- Node feature propagation
  - Spectral CNN
  - ChebyNet
  - GNN
  - GAT
Graph data and notation

• Structured data: Graph \( G = (V, E) \) where \( V \) is the node set and \( E \) is the edge set. \( |V| = n \)

• Extra information:
  • Node features/labels
  • Edge features/labels

Node feature learning

• Input: \( G = (V, E) \) (or (weighted) adjacency matrix \( A \in \mathbb{R}^{n \times n} \))

• Output:
  • feature representations on each node
  • \( Z \in \mathbb{R}^{n \times d}; d < n \)

• Categories:
  • Factorization based methods
  • Random walk based methods
  • Other sampling-based methods
A simple idea

• How to assign a real number to each vertex in the graph? (learn 1-d features)
  • Vertices with linkages have similar values
    \[
    \min_z \sum_{(i,j) \in E} A_{ij} (z_i - z_j)^2
    \]
  • We need to exclude the trivial solution \( z = 1 \). We require the solution to be orthogonal to 1:
    \[
    1^T z = 0
    \]

A simple idea

• A more compact representation:
  • Let \( L \) be the Laplacian matrix of \( A \)
    \[
    L = D - A
    \]
  • where \( D \) is the degree matrix:
    \[
    D_{ii} = \sum_j A_{ij}
    \]
  • The original problem is equivalent to
    \[
    \min_{z; 1^T z = 0} z^T L z
    \]
  • The eigenvector corresponding to the second smallest eigenvalue!
A simple idea

- We can generalize it to the whole eigen-space:
  \[
  \min_{\text{Orthonomal } Z} \left| z_i - z_j \right|^2 A_{i,j} = \text{Tr}(Z^T L Z)
  \]

- where Tr(·) is the trace operator (sum of the diagonal)

Some interesting examples

- Examples from [yale lecture]
- Ring-shape graph
- Grid graph
- Platonic solid

(a) The ring graph on 9 vertices. (b) The eigenvector for k = 2.
Factorization based methods

• Common framework:
  • Define a decoding algorithm Dec for edges based on embeddings:
  • Optimize for
    \[
    \min_Z \text{loss}(\text{Dec}(z_i, z_j), A_{i,j})
    \]
  • A popular choice for the loss function is the squared loss
  • Our simple idea (Laplacian eigenmaps)
    \[
    \min_{\text{Orthonormal } Z} |z_i - z_j|^2 A_{i,j}
    \]

Factorization based methods

• Laplacian eigenmaps:
  • \[
  \min_{\text{Orthonormal } Z} |z_i - z_j|^2 A_{i,j} = \text{Tr}(Z^T LZ)
  \]
• Graph factorization:
  • \[
  \min_z \sum_{i,j} |z_i^T z_j - A_{i,j}|
  \]
• GraRep (CIKM15):
  • \[
  \min_z \sum_{i,j} |z_i^T z_j - \log \left( \frac{A_{i,j}^k}{\sum_j A_{i,j}^k} \right) + \log(\beta) |
  \]
  • Concatenate features of different \( k \) values
  • Analogy of the word2vec negative sampling
Random walk based methods

- Factorization-based methods care about the global edge information
- Sometimes efficiency also matters (not using the full graph)
- How to represent a graph without using the full adjacency matrix:
  - Sampling paths from the graph!

Random walk based methods

- Paths ↔ Sentences
- Nodes ↔ Words
- Recall how word embeddings are learned [CS224 Stanford]:

![Diagram 1](image1)
![Diagram 2](image2)
Random walk based methods

• DeepWalk (KDD14):

```
Algorithm 1 DeepWalk(G, w, d, γ, t)
Input: graph G(V, E)
    window size w
    embedding size d
    walks per vertex γ
    walk length t
Output: matrix of vertex representations \( \Phi \in \mathbb{R}^{V \times d} \)
1: Initialization: Sample \( \Phi \) from \( \mathcal{U}^{V \times d} \)
2: Build a binary Tree \( T \) from \( V \)
3: for \( i = 0 \) to \( γ \) do
4:   \( \mathcal{O} = \) Shuffle(\( V \))
5:   for each \( v_i \in \mathcal{O} \) do
6:     \( \mathcal{W}_{v_i} = \) RandomWalk(\( G, v_i, t \))
7:     SkipGram(\( \Phi, \mathcal{W}_{v_i}, w \))
8: end for
9: end for
```

Random walk based methods

• DeepWalk adopts a uniform sampling strategy
• We may want a more diverse path
  • A path should not always loop around (reducing probability of turning back)
  • Hubs should not always act as the sink of all paths
Random walk based methods

- Node2Vec (KDD16):
  - Modified transition matrix
  - Reweight nodes with common neighbors/replicate nodes

Other sampling based methods

- GraphGAN (AAAI18):
  - Intuition: (true) samplings of nodes from neighborhood should be close to a learnt generator sampler

Figure 1: Illustration of GraphGAN framework.
Figure 2: Illustration of the random walk procedure in node2vec. The walk just transitioned from t to v and is now evaluating its next step out of node v. Edge labels indicate search biases $\alpha$. 
Other sampling based methods

• GAN-like loss:

\[
\min_{\theta_G} \max_{\theta_D} V(G, D) = \frac{1}{\alpha} \mathbb{E}_{x \sim p_{data}(v)} \left[ \log D(v, v_x; \theta_D) \right] + \mathbb{E}_{x \sim \tilde{p}(v; \theta_G)} \left[ \log (1 - D(v, v_x; \theta_D)) \right].
\]

• Discriminator design:

\[
D(v, v_x) = \sigma(d_1 d_2) = \frac{1}{1 + \exp(-d_1 d_2)}.
\]

• Generator design (hierarchical):

\[
p_c(v | v) = \frac{\exp(g_{v_x} g_v)}{\sum_{v_x \in \mathcal{N}(v)} \exp(g_{v_x} g_v)}, \quad G(v | v) = (\prod_{j=1}^{m} p_c(v_{t_j} | v_{t_{j-1}})) \cdot p_c(v_{t_m} | v_{t_{m-1}}),
\]

Node feature propagation

• Input: \( G = (V, E) \) (or (weighted) adjacency matrix \( A \in \mathbb{R}^{n \times n} \), node-wise features \( X \in \mathbb{R}^{n \times d} \)

• Output:
  • New convolved feature representations on each node
  • \( Z \in \mathbb{R}^{n \times d'} \)

• Downstream tasks:
  • Vertex classification
  • Link prediction
Node feature propagation

- Spectral CNN (ICLR14): generalization from normal CNN
- ChebNet (NIPS16): faster computation
- Graph convolution network (GCN) (ICLR17): first-order approximation
- Graph attentional networks (GAT) (ICLR18): attention mechanism

---

Figure 1: Undirected Graph $G = (\Omega, W)$ with two levels of clustering. The original points are drawn in gray.
Node feature propagation

• Spectral CNN (ICLR14)
  • Spectral generalization

\[ x_{k+1,j} = h \left( \sum_{i=1}^{f_{k-1}} F_{k,i,j} V^T x_{k,i} \right) \quad (j = 1 \ldots f_k) , \]

• where \( V \) is the eigenvector matrix for the Laplacian matrix \( L = V \Sigma V^T \)
• Computation: \( O(n^2 f_{k-1}) \)

Node feature propagation

• ChebNet (NIPS16):
  • Key idea: diagonal matrix \( F \) could be approximated by polynomials of original diagonal matrix \( \Sigma \) (e.g. order \( r \))
  • Specifically, use Chebyshev polynomials:
    \[ T_0(x) = 1, T_1(x) = x, T_j(x) = 2T_{j-1}(x) - T_{j-2}(x) \]
    • Recursive computable, orthonormal basis in \([-1,1]\)
  
  Then

\[ x_{k+1,j} = h \left( \sum_{r \in I} \alpha_{r,i} T_{r,i}(L) x_{k,i} \right) \]

• Computation: \( O(r \text{ nnz}(L) f_{k-1}) \)
Node feature propagation

- Graph Neural Network (GNN) (ICLR17):
  - Key idea: even aggressive approximation by keeping the first order polynomials
  - For simplicity suppose $i = 1$
    $$\alpha_0 x_k + \alpha_1 (D - A)x_k$$
  - normalized version
    $$\alpha_0 x_k + \alpha_1 (I - D^{-\frac{1}{2}}AD^{-\frac{1}{2}})x_k$$
  - Make $\alpha_0 + \alpha_1 = -\alpha_1 = \alpha$, we get
    $$\alpha(I + D^{-\frac{1}{2}}AD^{-\frac{1}{2}})x_k$$

Node feature propagation

- Graph attention networks (GAT) (ICLR18):
  - Key idea: layer-wise aggregation is the weighted attention sum of previous layers
  - Input: features $h = \{h_1, h_2, ..., h_N\}$
  - Output: features $h' = \{h'_1, h'_2, ..., h'_N\}$
    $$e_{ij} = a(W\tilde{h}_i, W\tilde{h}_j)$$
    $$\alpha_{ij} = \text{softmax}_j(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k \in N_i}\exp(e_{ik})}.$$  
  - Multi-head:
    $$\tilde{h}'_i = \sum_{k=1}^{K} \sigma(\sum_{j \in N_i} \alpha_{ij}^k W^k \tilde{h}_j)$$
How to make a prediction over the whole graph?

• A Conv, Combine and Readout pipeline

\[ a_v^{(k)} = \text{AGGREGATE}^{(k)} \left( \{ h_u^{(k-1)} : u \in N(v) \} \right), \quad h_v^{(k)} = \text{COMBINE}^{(k)} \left( h_v^{(k-1)}, a_v^{(k)} \right) \]

\[ h_G = \text{READOUT}(\{ h_v^{(K)} : v \in G \}). \]

Current progress

• Different modeling techniques (factorization/sampling)
• Balance of local information aggregation and computation (graph convolution)
• Interesting topics not covered:
  • Graph kernels
  • Wavelet analogy on graphs
References

- Representation Learning on Graphs: Methods and Applications
- Graph Embedding Techniques, Applications, and Performance: A Survey
- GraRep- Learning Graph Representations with Global Structural Information
- DeepWalk: online learning of social representations
- node2vec: Scalable Feature Learning for Networks
- GraphRNN: Generating Realistic Graphs with Deep Auto-regressive Models
- Spectral networks and locally connected networks on graphs
- Convolutional neural networks on graphs with fast localized spectral filtering
- Semi-supervised classification with graph convolution networks
- Graph attention networks
- Complex embeddings for simple link prediction
- Adversarial Attack on Graph Structured Data