Graph Embedding Tutorial

Maosen Zhang

Graph Analysis

- Modeling the interactions between entities
- Tasks:
  - Node classification
  - Link Prediction: Predicting missing links or links that are likely to occur in the future
  - Clustering: Find subsets of similar nodes and group them together
  - Visualization
- Operation:
  - On the original graph adjacency matrix
  - On a derived vector space

Challenges of Graph Embedding

- Choice of property
  - Choosing the property of the graph which the embedding should preserve
- Scalability
  - Most real networks are large and contain millions of nodes and edges
- Dimensionality of the embedding
  - higher number of dimensions may increase the reconstruction precision but will have high time and space complexity

Definitions

- Graph $G(V, E)$:
  - vertices $V = \{v_1, \ldots, v_n\}$
  - Edges $E = \{e_{ij}\}_{i,j=1}^{n}$
- Adjacency Matrix S: non-negative weights associated with each edge: $s_{ij} \geq 0$
  - $s_{ij} = 0$ if v_i and v_j are not connected
  - Higher, more similar
  - For undirected weighted graphs: $s_{ij} = s_{ji}$

Definitions

- First-order Proximity: weights of edges between nodes s_{ij}
- Second-order Proximity:
  - $\boldsymbol{s_i} = [s_{i1}, \dots, s_{in}]$
  - Second-order proximity between v_i and v_j determined by similarity of $\boldsymbol{s_i}$ and $\boldsymbol{s_j}$
- Graph Embedding:
  - Mapping $f: v_i \rightarrow y_i \in \mathbb{R}^d, \forall i \in [n]$
  - $d \ll |V|$

Approaches

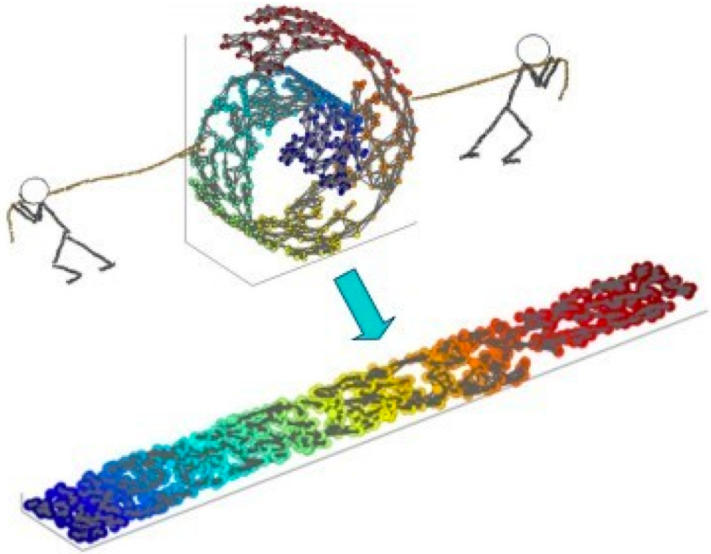
- Factorization based methods
- Random walk based methods
- Deep learning based methods

Factorization based Methods

- Represent the connections between nodes in the form of matrix
- Factorize the matrix to obtain the embedding
- Matrices:
  - Adjacency matrix
  - Laplacian matrix
  - Node transition probability matrix
  - Katz similarity matrix
- Factorization approaches:
  - positive semidefinite (e.g. the Laplacian matrix): eigenvalue decomposition
  - unstructured matrices: gradient descent methods

Locally Linear Embedding (LLE)

- Dimensionality reduction
  - While keeping local and global structures
- X_i(i=1,…,n) are input high-dimensionality data
- Assumes every node is a linear combination of its neighbors in the embedding space
- Minimize $J(W) = \sum_i |X_i - \sum_j W_{ij} X_j|^2$: compute W
  - $\sum_j W_{ij} = 1$
  - W_{ij} higher, X_i and X_j more similar
- Minimize $\phi(Y) = \sum_i |Y_i - \sum_j W_{ij} Y_j|^2$
  - Y_i is the low-dimension embedding of X_i



Locally Linear Embedding (LLE)

- Assumes every node is a linear combination of its neighbors in the embedding space
- In graph embedding:
  - W is *normalized* adjacency matrix: $\sum_j W_{ij} = 1$; Y is embedding vectors
- Minimize $\phi(Y) = \sum_i |Y_i - \sum_j W_{ij} Y_j|^2 = \sum_i \left|\sum_j W_{ij}(Y_i - Y_j)\right|^2$
  - Remove degenerate solutions: $\frac{1}{\mathrm{N}} \sum_{i=1}^{n} Y_i Y_i^T = I$
  - Remove translational invariance: $\sum_i Y_i = 0$
- Let $M = (I - W)^T (I - W)$
- M's eigenvectors: $(m_2, m_3, \ldots, m_{d+1})$ -> Y
- https://www.cnblogs.com/pinard/p/6266408.html?utm_source=itdadao&utm_medium=referral
- https://github.com/ArrowLuo/LLE_Algorithm/blob/master/%E5%B1%80%E9%83%A8%E7%BA%BF%E6%80%A7%E5%B5%8C%E5%85%A5%EF%BC%88Locally%20linear%20embedding%EF%BC%89.pdf

Laplacian Eigenmaps

- Keep the embeddings of two nodes close when the weight W_{ij} is high
  - W is adjacency matrix
- Minimize $\phi(Y) = \frac{1}{2}\sum_{i,j}\left|Y_i - Y_j\right|^2 W_{ij} = tr(Y^T L Y)$
  - constraint: $Y^T D Y = I$
  - Diagonal matrix D: $D_{ii} = \sum_{j=1}^{n} W_{ij}$
  - $L = D - W$: Laplacian matrix
- Lagrange multiplier method
- Take the eigenvectors corresponding to the d smallest eigenvalues of the normalized Laplacian $L_{norm} = D^{-1/2} L D^{-1/2}$
- https://blog.csdn.net/qrlhl/article/details/78066994

Graph Factorization

- Factorize adjacency matrix W
- Minimize $\phi(Y,\lambda) = \frac{1}{2}\sum_{(i,j)\in E}\left(W_{ij} - < Y_i, Y_j >\right)^2 + \frac{\lambda}{2}\sum_i ||Y_i||^2$
  - λ: regularization coefficient
- Stochastic gradient descent

GraRep

- Capture higher order graph proximity
- Probability transition matrix $A = D^{-1}W$
  - Diagonal matrix D: $D_{ii} = \sum_{j=1}^{n} W_{ij}$
  - $A_{ij} = \frac{W_{ij}}{\sum_{j=1}^{n} W_{ij}}$
- K-step probability transition matrix: A^k: $p(c|w) = A_{w,c}^k$

GraRep

- Let $X_{i,j}^k = \log(\frac{A_{i,j}^k}{\sum_p A_{p,j}^k})$
- $\left[U^k, \Sigma^k, \left(V^k\right)^T\right] = SVD(X^k)$
- Minimize $\left|X^k - Y_s^k {Y_t^k}^T\right|^2$
  - $Y_s^k = U_d^k \left(\Sigma_d^k\right)^{\frac{1}{2}}, Y_t^k = \left(\Sigma_d^k\right)^{\frac{1}{2}} {V_d^k}^T$
  - $X^k \approx X_d^k = U_d^k \Sigma_d^k \left(V_d^k\right)^T$
- $Y_s^k = U_d^k \left(\Sigma_d^k\right)^{\frac{1}{2}}$
- Concatenate: $Y = [Y_s^1, Y_s^2, \dots, Y_s^k]$

GraRep & HOPE

- Skip-gram: special case of GraRep
- HOPE: change X^k to S
  - General similarity matrix
- SVD to obtain embedding

Approaches

- Factorization based methods
- Random walk based methods
- Deep learning based methods

Random Walk based Methods

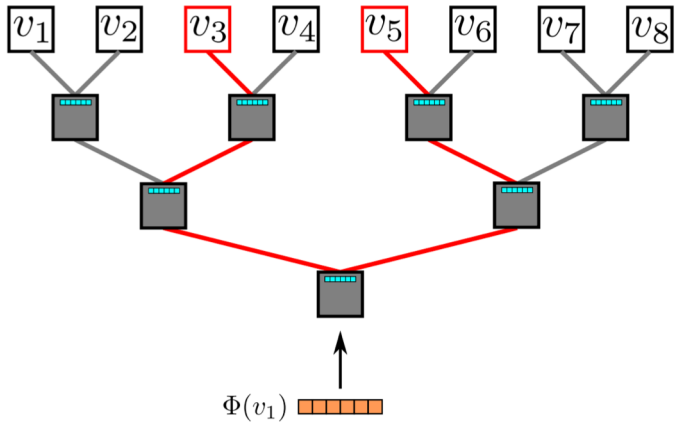
- Random walk rooted at vertex v: W_v
  - Stochastic process with random variables $W_v^1, W_v^2, \ldots W_v^k$
  - W_v^{k+1} is a vertex chosen at random from the neighbors of vertex W_v^k

DeepWalk

- Random walk generator
  - Takes a graph G and samples uniformly a random vertex v_i as the root of the random walk W_{v_i}
  - Sample uniformly from the neighbors of the last vertex visited, until the maximum length (t) reached.
- Update procedure
  - Skip-Gram
  - Minimize $-\log \Pr(\{v_{i-w}, \dots, v_{i-1}, v_{i+1}, \dots, v_{i+w}\} | \Phi(v_i))$
  - Remove order constraint: $-\log \sum_{u_k \in W_v[j-w, j+w]} \Pr(u_k | \Phi(v_j))$
- Stochastic gradient descent

DeepWalk

- Hierarchical Softmax: accelerate from $O(|V|)$ to $O(\log|V|)$
- Assign the vertices to the leaves of a binary tree:
  - Maximizing the probability of a specific path in the tree
  - Huffman coding
- Path to vertex u_k: $b_0, b_1, \ldots, b_{\lceil \log|V| \rceil}$
- $\Pr(u_k|\Phi(v_j)) = \prod_{l=1}^{\lceil \log|V| \rceil} \Pr(b_l|\Phi(v_j))$
- $\Pr(b_l|\Phi(v_j)) = \frac{1}{1+\exp(-\Phi(v_j)^T \Psi(b_l))}$
- $\Psi(b_l)$: representation assigned to tree node b_l's parent



DeepWalk

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 $\Phi \in \mathbb{R}^{|V| \times d}$

Initialization: Sample Φ from $\mathcal{U}^{|V| \times d}$
Build a binary Tree T from V
for $i = 0$ to γ **do**
  $\mathcal{O} = \text{Shuffle}(V)$
  for each $v_i \in \mathcal{O}$ **do**
    $\mathcal{W}_{v_i} = RandomWalk(G, v_i, \text{t})$
    SkipGram(Φ, $\mathcal{W}_{v_i}$, w)
  end for
end for

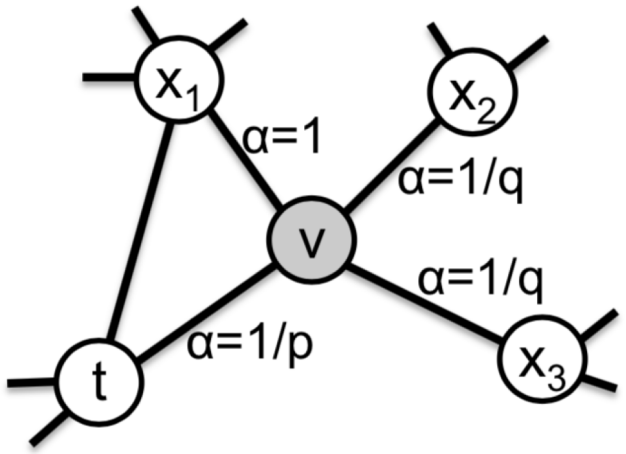
DeepWalk

Algorithm 2 SkipGram(Φ, $\mathcal{W}_{v_i}$, w)

1: **for each** $v_j \in \mathcal{W}_{v_i}$ **do**
2: **for each** $u_k \in \mathcal{W}_{v_i}[j-w : j+w]$ **do**
3: $J(\Phi) = -\log \Pr(u_k \mid \Phi(v_j))$
4: $\Phi = \Phi - \alpha * \frac{\partial J}{\partial \Phi}$
5: **end for**
6: **end for**

Node2Vec

- Biased (second order) random walk:
- In $t \to v \to x$:
- Transition probability (unnormalized)



- $\alpha_{pq}(t,x) = \begin{cases} \frac{1}{p}, if\ d_{tx} = 0 \\ 1, if\ d_{tx} = 1 \\ \frac{1}{q}, if\ d_{tx} = 2 \end{cases}$

- d_{tx}: shortest path distance between nodes t and x
- p,q: return parameter & in-out parameter
- Tradeoff (Interpolate) between DFS and BFS
  - q>1: more BFS (biased to closer nodes)
  - q<1: more DFS;

Node2Vec

- SGD
- Negative Sampling
- $\Pr\left(v_j \middle| \Phi(v_i)\right) = \frac{\exp\left(\Phi(v_j)^T \Phi(v_j)\right)}{\sum_v \exp\left(\Phi(v_j)^T \Phi(v)\right)}$

Algorithm 1 The *node2vec* algorithm.

LearnFeatures (Graph $G = (V, E, W)$, Dimensions d, Walks per node r, Walk length l, Context size k, Return p, In-out q)
π = PreprocessModifiedWeights(G, p, q)
$G' = (V, E, \pi)$
Initialize $walks$ to Empty
for $iter = 1$ **to** r **do**
  for all nodes $u \in V$ **do**
    $walk$ = node2vecWalk(G', u, l)
    Append $walk$ to $walks$
f = StochasticGradientDescent(k, d, $walks$)
return f

node2vecWalk (Graph $G' = (V, E, \pi)$, Start node u, Length l)
Inititalize $walk$ to $[u]$
for $walk\_iter = 1$ **to** l **do**
  $curr = walk[-1]$
  V_{curr} = GetNeighbors($curr, G'$)
  s = AliasSample(V_{curr}, π)
  Append s to $walk$
return $walk$

Node2Vec

- Trade-off between DFS and BFS
- Community structure and structural equivalence between nodes
- Higher quality, more informative embeddings

Hierarchical Representation Learning for Networks

- DeepWalk and node2vec initialize randomly
- Graph coarsening:
  - Collapse related nodes together into “supernodes”
- Generates embedding of the coarsest graph
- Initializes the node embeddings of the refined graph with the learned embedding
- Propagates such embeddings through the hierarchy to obtain the embeddings of the original graph
  - Use DeepWalk, node2vec, etc.

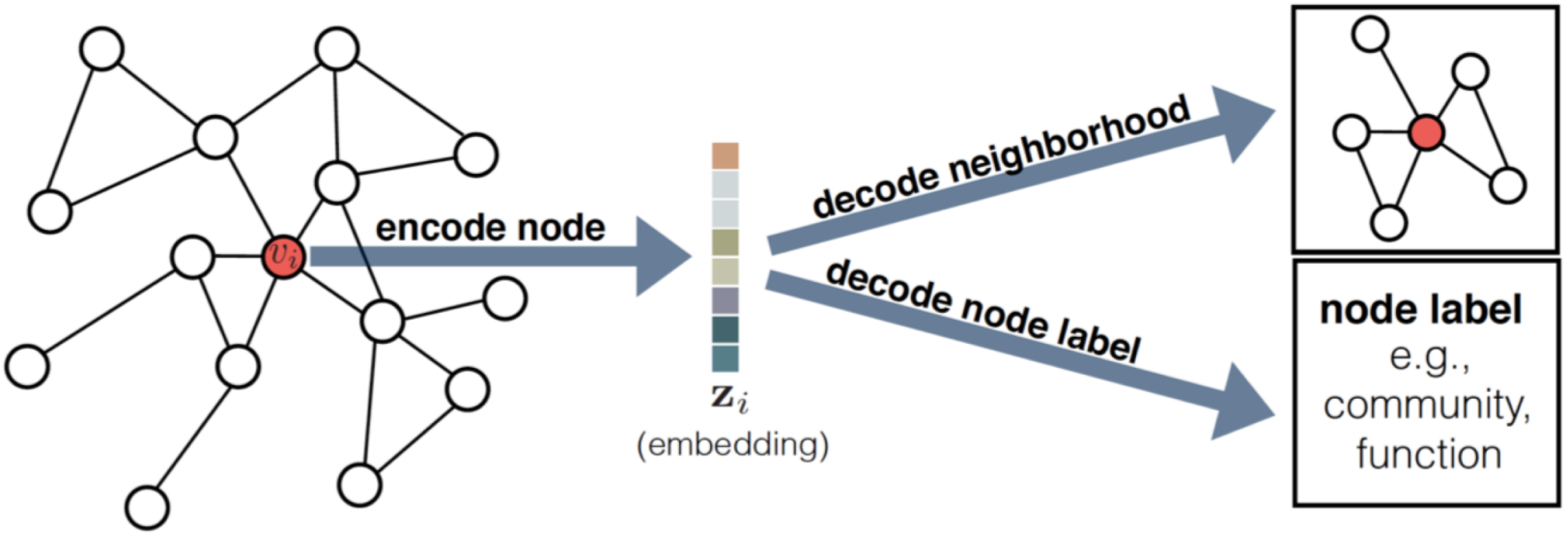
LINE: Large-scale Information Network Embedding

- Combines two objectives that optimize first-order and second-order proximity respectively
- Defines two joint probability distributions for each pair of vertices:
- Adjacency matrix & embedding
- Minimize the KL divergence of these two distributions
- First-order:
- $p_1(v_i, v_j) = \frac{1}{1+\exp(-<u_i, u_j>)}$
- $\widehat{p_1}(v_i, v_j) = \frac{W_{ij}}{\sum_{(i,j)\in E} W_{ij}}$
- $O_1 = KL(\widehat{p_1}, p_1) = -\sum_{(i,j)\in E} W_{ij} \log p_1(v_i, v_j)$

LINE: Large-scale Information Network Embedding

- Second-order:
- $p_2(v_j|v_i) = \frac{\exp(u_j^T u_i)}{\sum_k \exp(u_j^T u_i)}$
- $\widehat{p_2}(v_j|v_i) = \frac{W_{ij}}{d_i}$
- $O_2 = \sum_i d_i KL(\widehat{p_2}(.|v_i), p_2(.|v_i))$
- $O_2 = -\sum_{(i,j)\in E} W_{ij} \log p_2(v_j|v_i)$
- Jointly train the two objective functions

An encoder-decoder perspective



An encoder-decoder perspective

1. A **pairwise proximity function** $s_{\mathcal{G}} : \mathcal{V} \times \mathcal{V} \rightarrow \mathbb{R}^+$, defined over the graph, $\mathcal{G}$. This function measures how closely connected two nodes are in $\mathcal{G}$.
2. An **encoder function**, ENC, that generates the node embeddings. This function contains a number of trainable parameters that are optimized during the training phase.
3. A **decoder function**, DEC, which reconstructs pairwise proximity values from the generated embeddings. This function usually contains no trainable parameters.
4. A **loss function**, ℓ, which determines how the quality of the pairwise reconstructions is evaluated in order to train the model, *i.e.*, how $\text{DEC}(\mathbf{z}_i, \mathbf{z}_j)$ is compared to the true $s_{\mathcal{G}}(v_i, v_j)$ values.

An encoder-decoder perspective

Type	Method	Decoder	Proximity measure	Loss function (ℓ)
Matrix factorization	Laplacian Eigenmaps [4]	$\|\mathbf{z}_i - \mathbf{z}_j\|_2^2$	general	$\text{DEC}(\mathbf{z}_i, \mathbf{z}_j) \cdot s_{\mathcal{G}}(v_i, v_j)$
	Graph Factorization [1]	$\mathbf{z}_i^\top \mathbf{z}_j$	$\mathbf{A}_{i,j}$	$\|\text{DEC}(\mathbf{z}_i, \mathbf{z}_j) - s_{\mathcal{G}}(v_i, v_j)\|_2^2$
	GraRep [9]	$\mathbf{z}_i^\top \mathbf{z}_j$	$\mathbf{A}_{i,j}, \mathbf{A}_{i,j}^2, ..., \mathbf{A}_{i,j}^k$	$\|\text{DEC}(\mathbf{z}_i, \mathbf{z}_j) - s_{\mathcal{G}}(v_i, v_j)\|_2^2$
	HOPE [44]	$\mathbf{z}_i^\top \mathbf{z}_j$	general	$\|\text{DEC}(\mathbf{z}_i, \mathbf{z}_j) - s_{\mathcal{G}}(v_i, v_j)\|_2^2$
Random walk	DeepWalk [46]	$\frac{e^{\mathbf{z}_i^\top \mathbf{z}_j}}{\sum_{k \in \mathcal{V}} e^{\mathbf{z}_i^\top \mathbf{z}_k}}$	$p_{\mathcal{G}}(v_j \mid v_i)$	$-s_{\mathcal{G}}(v_i, v_j) \log(\text{DEC}(\mathbf{z}_i, \mathbf{z}_j))$
	node2vec [27]	$\frac{e^{\mathbf{z}_i^\top \mathbf{z}_j}}{\sum_{k \in \mathcal{V}} e^{\mathbf{z}_i^\top \mathbf{z}_k}}$	$p_{\mathcal{G}}(v_j \mid v_i)$ (biased)	$-s_{\mathcal{G}}(v_i, v_j) \log(\text{DEC}(\mathbf{z}_i, \mathbf{z}_j))$

Drawbacks of Direct Encoding Approaches

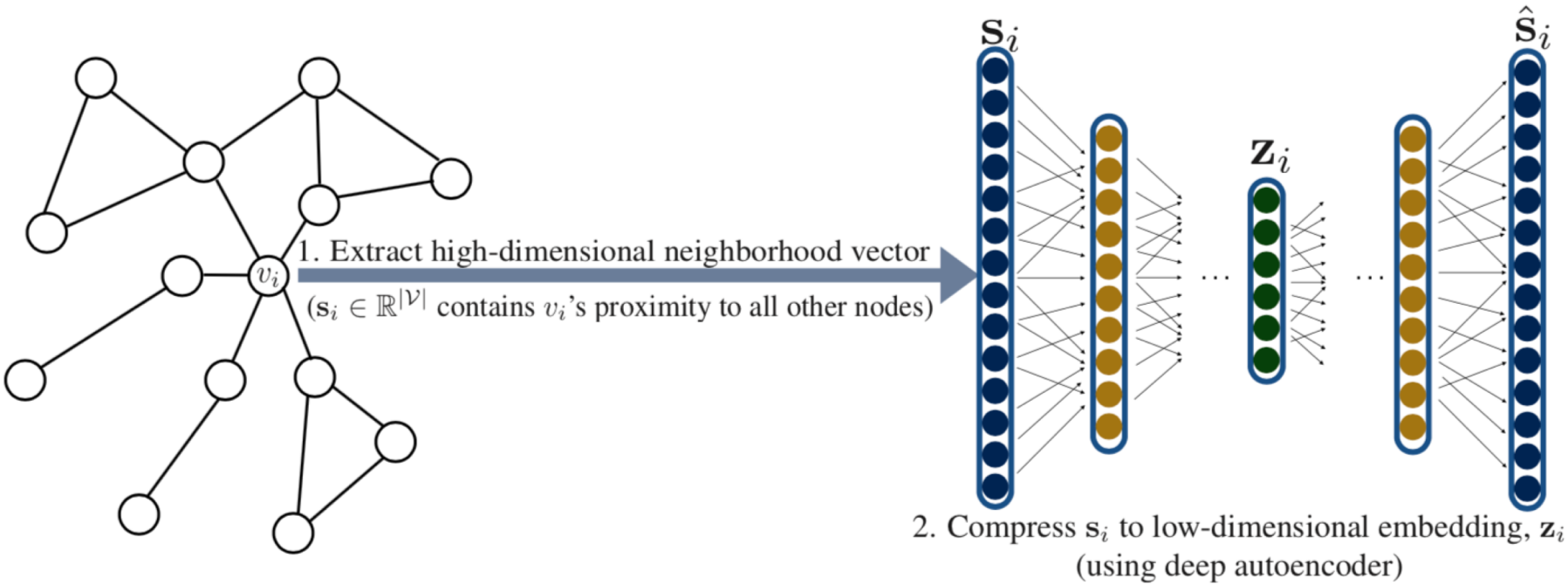
- No parameters are shared between: inefficient
  - parameter sharing can act as a regularization
  - the number of parameters necessarily grows as O(|V|)
- Fails to leverage node attributes during encoding
  - e.g., user profiles on a social network, node's position and role
- Can only generate embeddings for nodes that were present during the training phase

Approaches

- Factorization based methods
- Random walk based methods
- Deep learning based methods

Neighborhood Autoencoder Methods

- Use autoencoders to compress information about node's neighborhood
- Each node v_i associated with a neighborhood vector $s_i \in \mathbb{R}^{|V|}$
  - Corresponds to v_i's row in the matrix S (pairwise proximities)



Neighborhood Autoencoder Methods

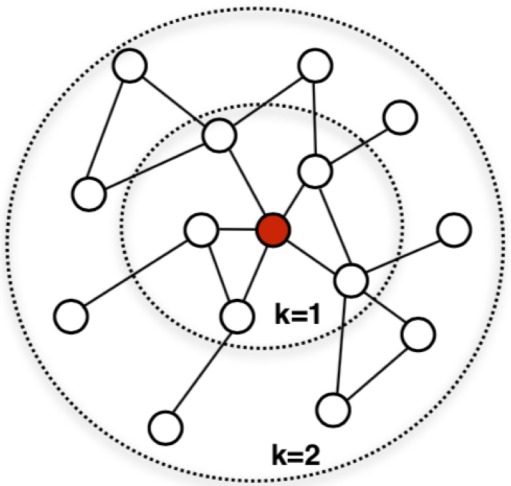
- Structural Deep Network Embeddings (SDNE)
  - S: adjacency matrix
- Deep Neural Graph Representations (DNGR)
  - S: pointwise mutual information of two nodes co-occurring on random walks
  - $PMI_{w,c} = \log \frac{\#(w,c)|D|}{\#(w)\#(c)}$
  - $|D| = \sum_w \sum_c \#(w,c)$
  - PPMI Matrix $X_{w,c} = \max(PMI_{w,c}, 0)$
- Minimize $\sum_i |DEC(z_i) - s_i|^2$

Neighborhood Autoencoder Methods

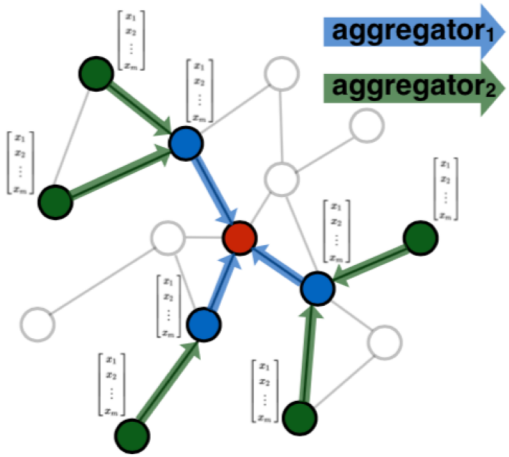
- Incorporate structural information about a node's local neighborhood directly into the encoder as a form of regularization
- Limitations
  - Input dimension is fixed at $|V|$:
  - The structure and size of the autoencoder is fixed
    - Cannot cope with evolving graphs, nor can they generalize across graph

Neighborhood Aggregation Methods

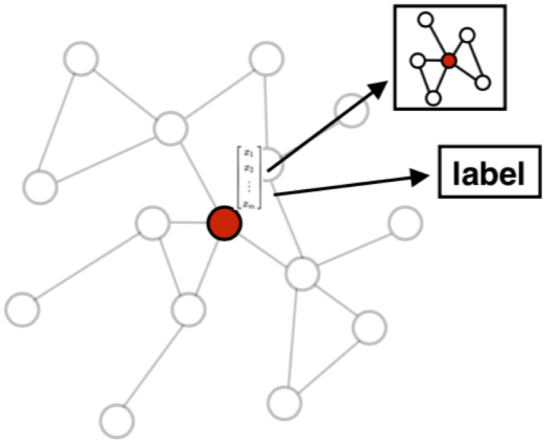
- Rely on node's local neighborhood, features and attributes



1. Collect neighbors



2. Aggregate feature information from neighbors



3. Decode graph context and/or label using aggregated information

Neighborhood Aggregation Methods

Algorithm 1: Neighborhood-aggregation encoder algorithm. Adapted from [28].

Input : Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$; input features $\{\mathbf{x}_v, \forall v \in \mathcal{V}\}$; depth K; weight matrices $\{\mathbf{W}^k, \forall k \in [1, K]\}$; non-linearity σ; differentiable aggregator functions $\{\text{AGGREGATE}_k, \forall k \in [1, K]\}$; neighborhood function $\mathcal{N} : v \rightarrow 2^{\mathcal{V}}$

Output: Vector representations $\mathbf{z}_v$ for all $v \in \mathcal{V}$

$\mathbf{h}_v^0 \leftarrow \mathbf{x}_v, \forall v \in \mathcal{V}$;
for $k = 1...K$ **do**
  for $v \in \mathcal{V}$ **do**
    $\mathbf{h}_{\mathcal{N}(v)}^k \leftarrow \text{AGGREGATE}_k(\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\})$;
    $\mathbf{h}_v^k \leftarrow \sigma\left(\mathbf{W}^k \cdot \text{COMBINE}(\mathbf{h}_v^{k-1}, \mathbf{h}_{\mathcal{N}(v)}^k)\right)$
  end
  $\mathbf{h}_v^k \leftarrow \text{NORMALIZE}(\mathbf{h}_v^k), \forall v \in \mathcal{V}$
end
$\mathbf{z}_v \leftarrow \mathbf{h}_v^K, \forall v \in \mathcal{V}$

Graph Convolutional Networks

- To be continued

Summary

Category	Year	Published	Method	Time Complexity	Properties preserved
Factorization	2000	Science[26]	LLE	$O(\|E\|d^2)$	1^{st} order proximity
	2001	NIPS[25]	Laplacian Eigenmaps	$O(\|E\|d^2)$	
	2013	WWW[21]	Graph Factorization	$O(\|E\|d)$	
	2015	CIKM[27]	GraRep	$O(\|V\|^3)$	$1-k^{th}$ order proximities
	2016	KDD[24]	HOPE	$O(\|E\|d^2)$	
Random Walk	2014	KDD[28]	DeepWalk	$O(\|V\|d)$	
	2016	KDD[29]	*node2vec*	$O(\|V\|d)$	$1-k^{th}$ order proximities, structural equivalence
Deep Learning	2016	KDD[23]	SDNE	$O(\|V\|\|E\|)$	1^{st} and 2^{nd} order proximities
	2016	AAAI[30]	DNGR	$O(\|V\|^2)$	$1-k^{th}$ order proximities
	2017	ICLR[31]	GCN	$O(\|E\|d^2)$	$1-k^{th}$ order proximities
Miscellaneous	2015	WWW[22]	LINE	$O(\|E\|d)$	1^{st} and 2^{nd} order proximities

Experiments

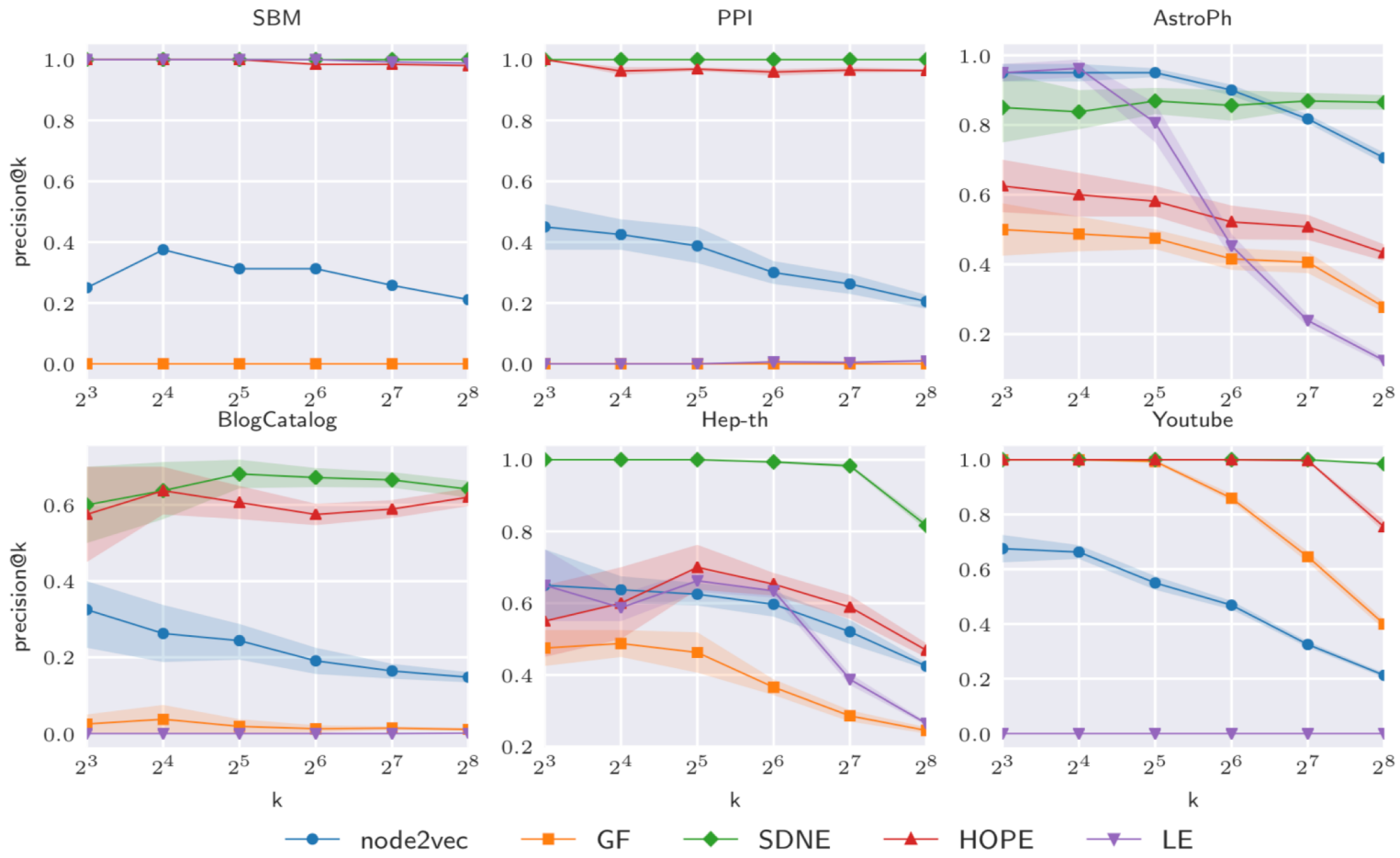


Figure 2: Precision@k of graph reconstruction for different data sets (dimension of embedding is 128).

Experiments

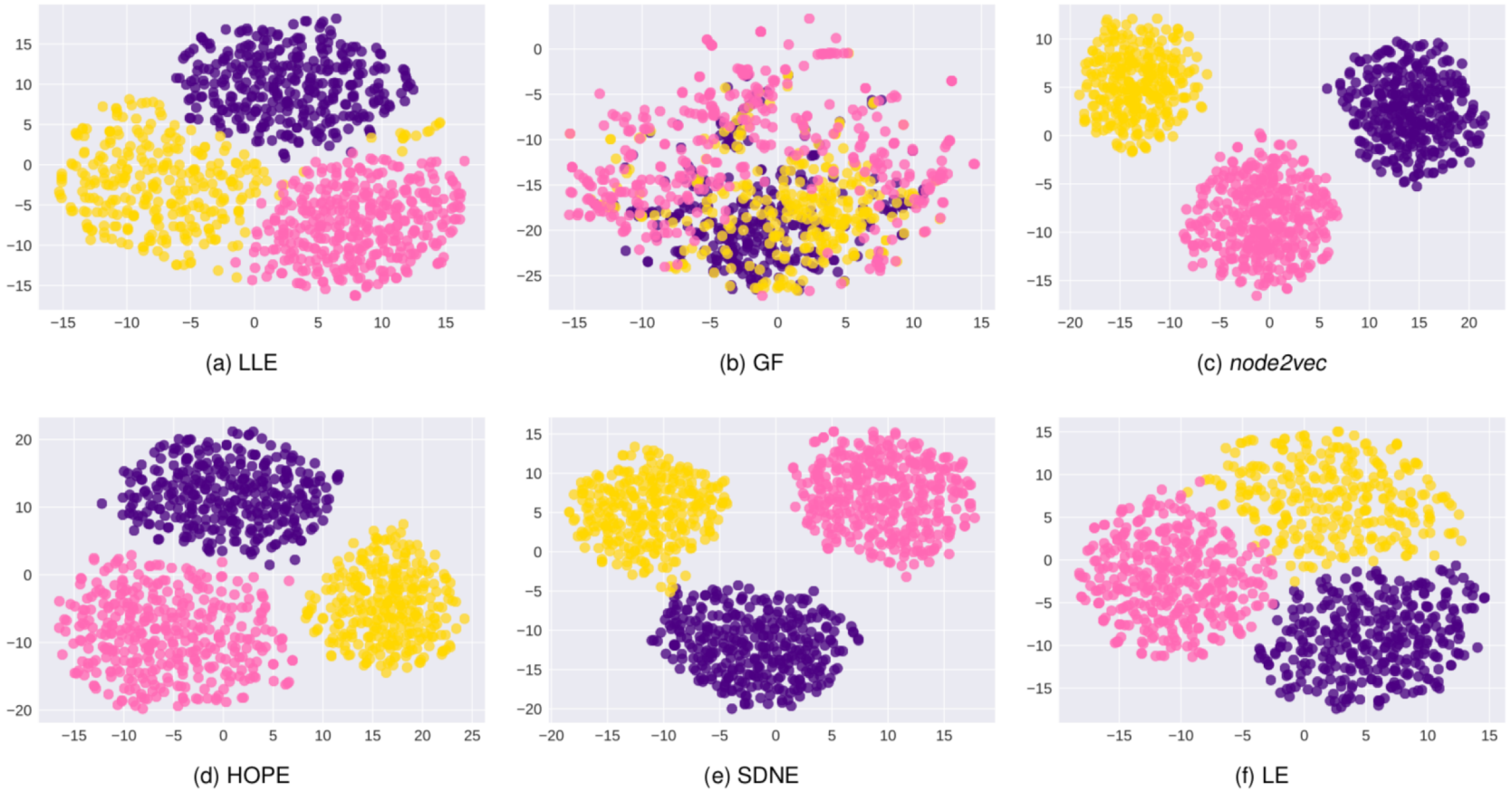


Figure 4: Visualization of SBM using t-SNE (original dimension of embedding is 128). Each point corresponds to a node in the graph. Color of a node denotes its community.

Experiments

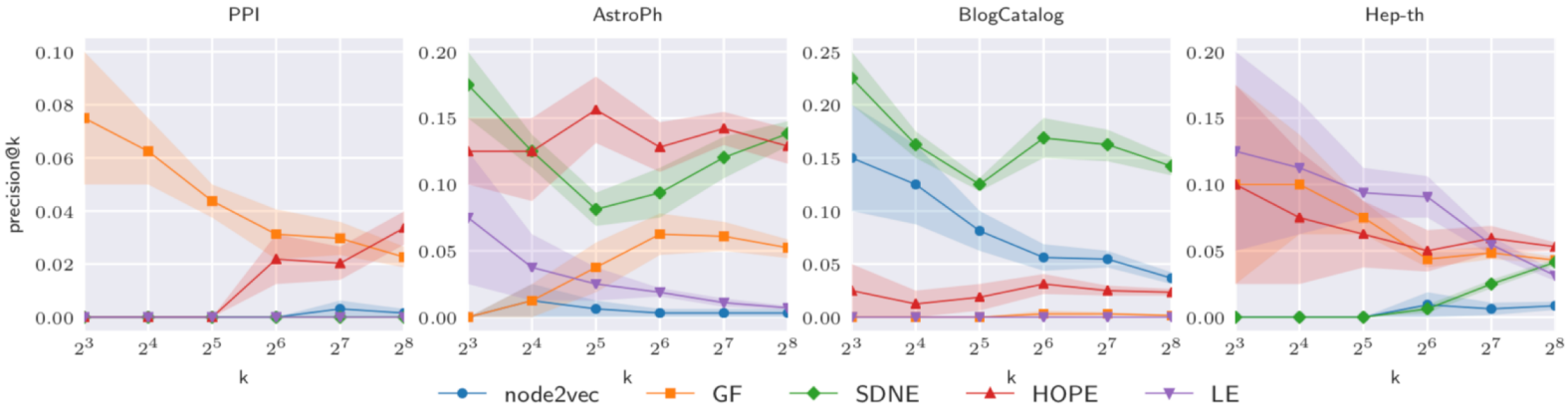


Figure 6: Precision@k of link prediction for different data sets (dimension of embedding is 128).

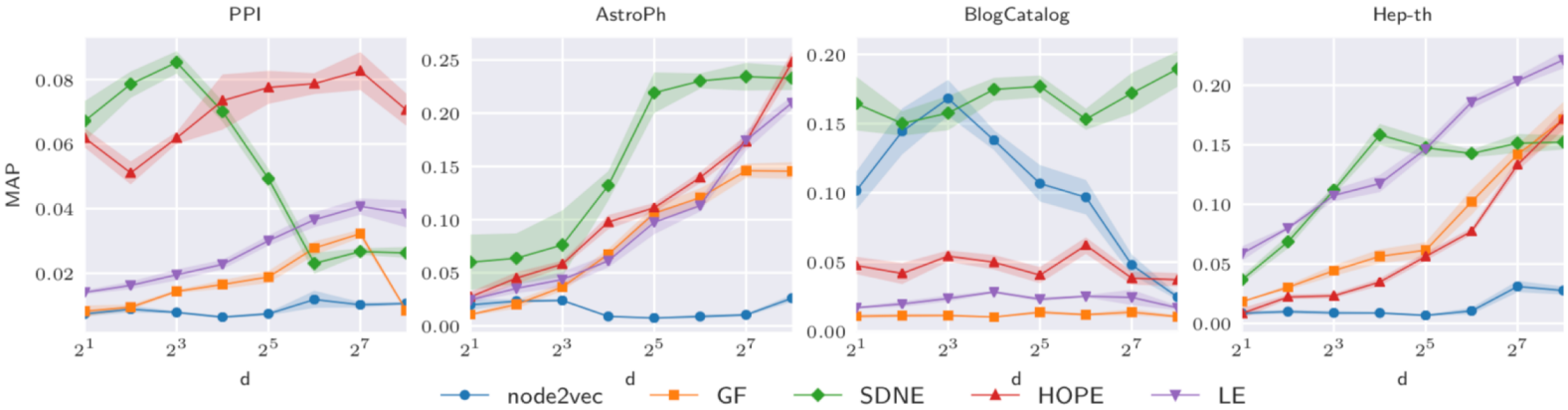


Figure 7: MAP of link prediction for different data sets with varying dimensions.

Experiments

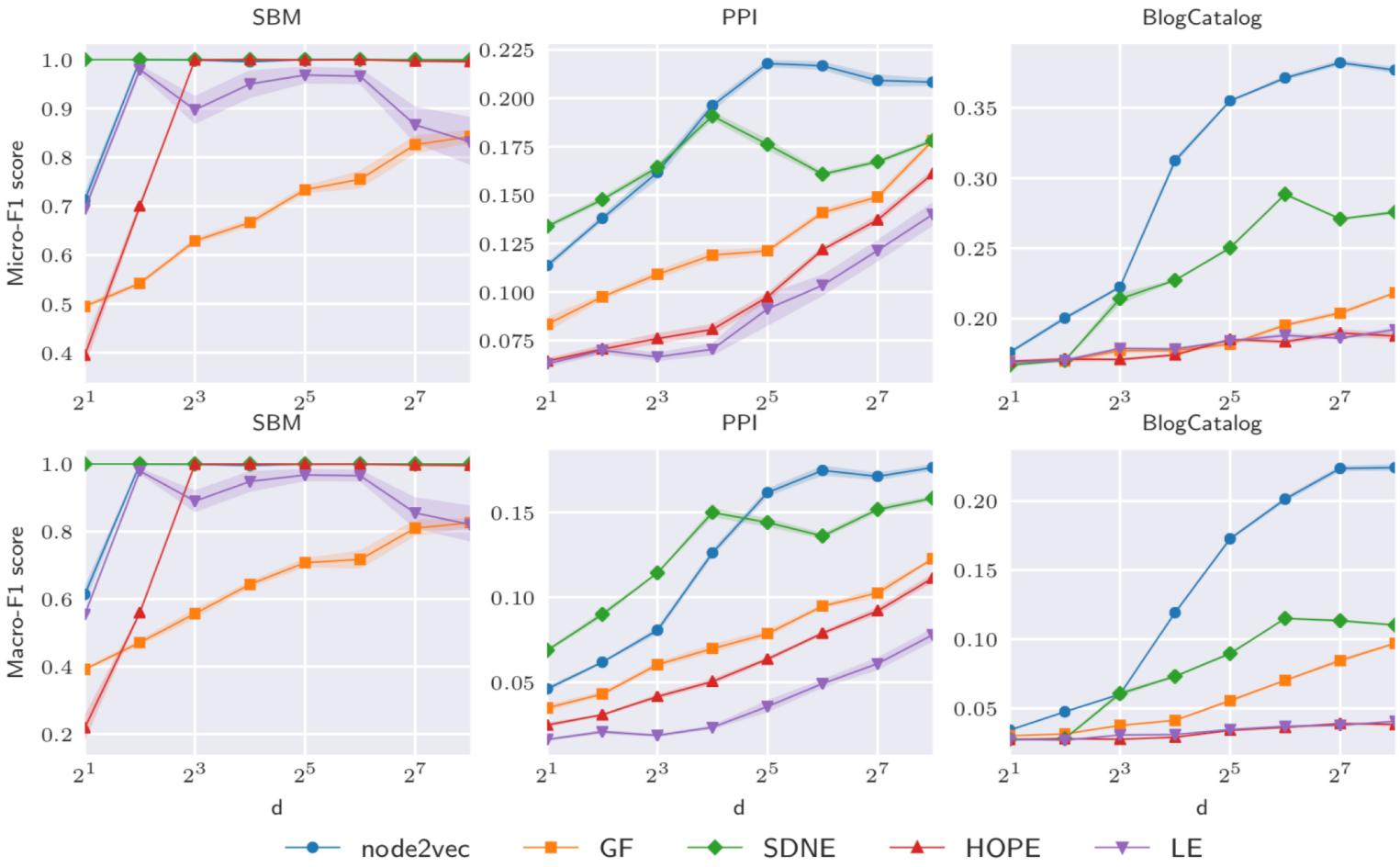


Figure 9: Micro-F1 and Macro-F1 of node classification for different data sets varying the number of dimensions. The train-test split is 50%.

References

- Palash Goyal and Emilio Ferrara. Graph Embedding Techniques, Applications, and Performance: A Survey
- William L. Hamilton, Rex Ying, Jure Leskovec. Representation Learning on Graphs: Methods and Applications
- https://www.cnblogs.com/pinard/p/6266408.html
- https://github.com/ArrowLuo/LLE_Algorithm/blob/master/%E5%B1%80%E9%83%A8%E7%BA%BF%E6%80%A7%E5%B5%8C%E5%85%A5%EF%BC%88Locally%20linear%20embedding%EF%BC%89.pdf
- https://blog.csdn.net/qrlhl/article/details/78066994
- S. T. Roweis, L. K. Saul, Nonlinear dimensionality reduction by locally linear embedding, Science 290 (5500) (2000) 2323–2326.
- A. Ahmed, N. Shervashidze, S. Narayanamurthy, V. Josifovski, A. J. Smola, Distributed large-scale natural graph factorization, WWW 2013.
- S. Cao, W. Lu, Q. Xu, Grarep: Learning graph representations with global structural information, CIKM 2015.
- B. Perozzi, R. Al-Rfou, S. Skiena, Deepwalk: Online learning of social representations, KDD 2014.
- A. Grover, J. Leskovec, node2vec: Scalable feature learning for networks, KDD 2016.
- J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, Q. Mei, Line: Large- scale information network embedding, WWW 2015.
- S. Cao, W. Lu, and Q. Xu. Deep neural networks for learning graph representations. AAAI 2016.
- T. N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks. ICLR 2017.