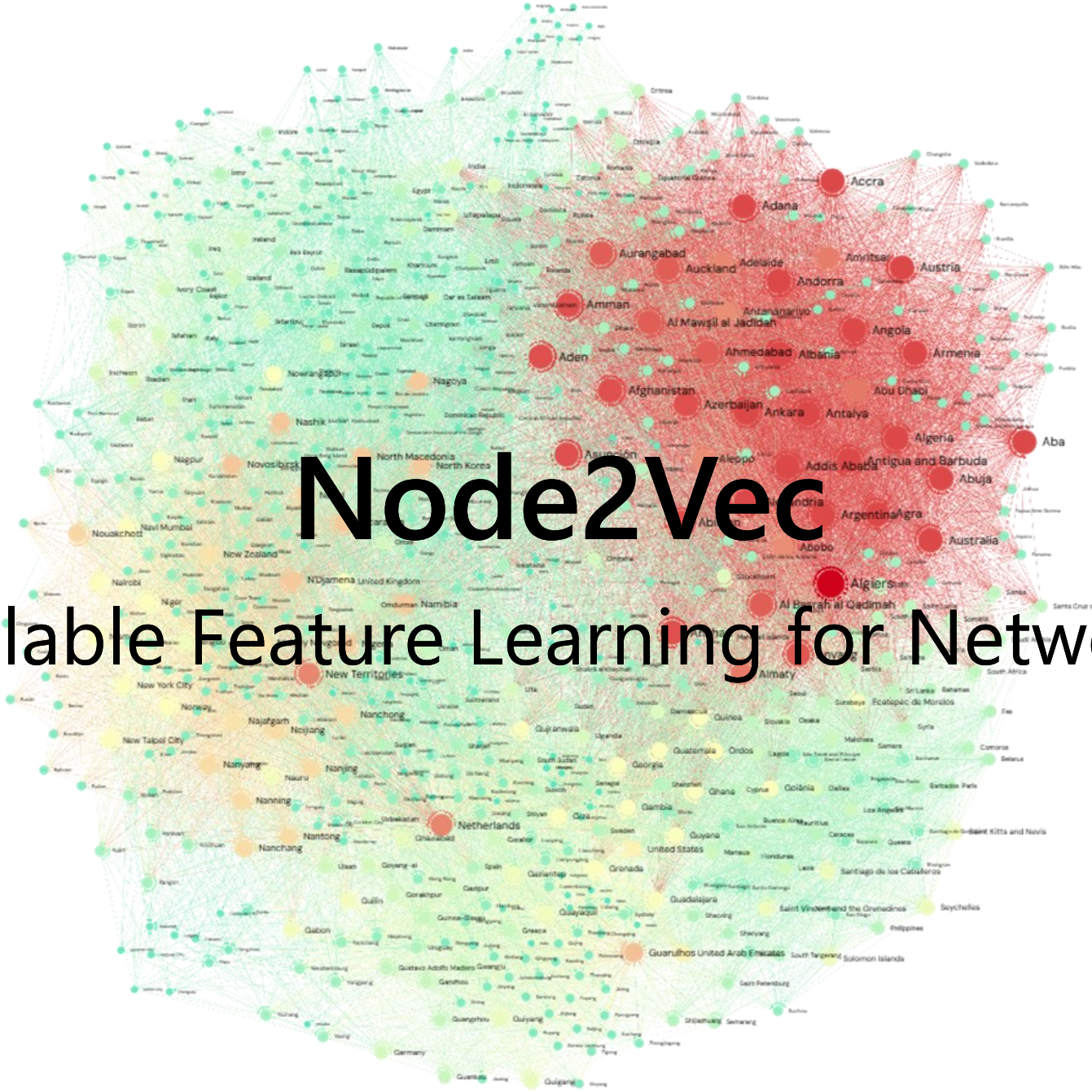


Node2Vec

Scalable Feature Learning for Networks

arXiv:1607.00653 (cs)

[Submitted on 3 Jul 2016]

node2vec: Scalable Feature Learning for Networks

Aditya Grover, Jure Leskovec

View PDF

Prediction tasks over nodes and edges in networks require careful effort in engineering features used by learning algorithms. Recent research in the broader field of representation learning has led to significant progress in automating prediction by learning the features themselves. However, present feature learning approaches are not expressive enough to capture the diversity of connectivity patterns observed in networks. Here we propose node2vec, an algorithmic framework for learning continuous feature representations for nodes in networks. In node2vec, we learn a mapping of nodes to a low-dimensional space of features that maximizes the likelihood of preserving network neighborhoods of nodes. We define a flexible notion of a node's network neighborhood and design a biased random walk procedure, which efficiently explores diverse neighborhoods. Our algorithm generalizes prior work which is based on rigid notions of network neighborhoods, and we argue that the added flexibility in exploring neighborhoods is the key to learning richer representations. We demonstrate the efficacy of node2vec over existing state-of-the-art techniques on multi-label classification and link prediction in several real-world networks from diverse domains. Taken together, our work represents a new way for efficiently learning state-of-the-art task-independent representations in complex networks.

Comments: In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016

Subjects: **Social and Information Networks (cs.SI)**; Machine Learning (cs.LG); Machine Learning (stat.ML)

Cite as: arXiv:1607.00653 **[cs.SI]**

(or arXiv:1607.00653v1 **[cs.SI]** for this version)

https://doi.org/10.48550/arXiv.1607.00653

Problem, Objective and Motivation

Problem Framing:

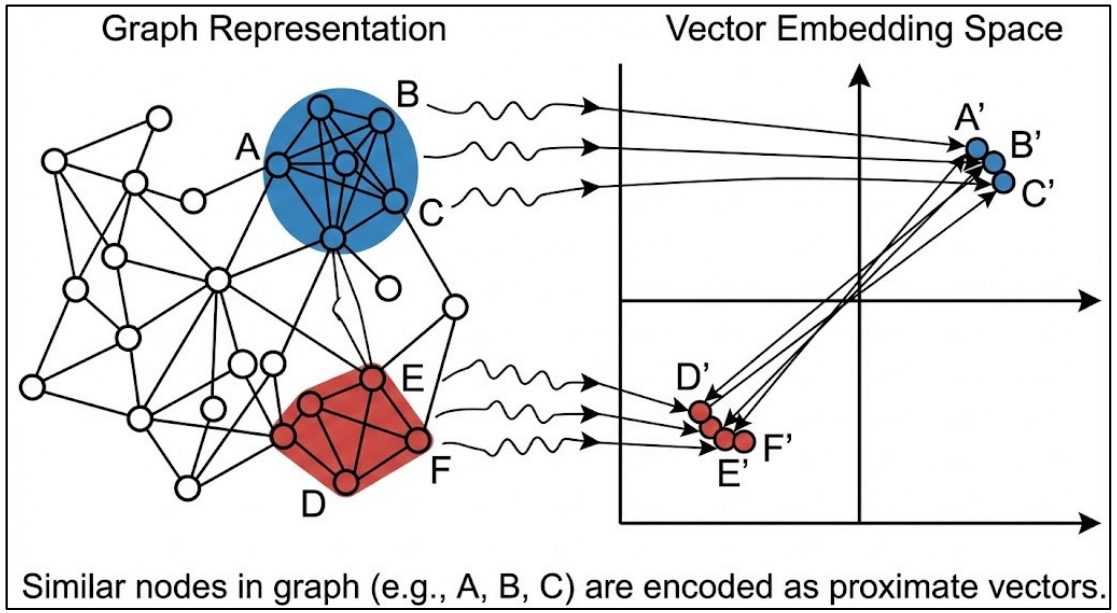
For a graph $\boldsymbol{G(V, E)}$, we want to learn an embedding function $\boldsymbol{f: V \rightarrow \mathbb{R}^d}$ where $\boldsymbol{d}$ is the dimension of the embedding space.

Objective:

Preserve structural roles and community structure of the nodes such that similar nodes in the graph have similar embeddings.

Motivation:

Create good feature representations for downstream ML tasks like multi-label classification and link-prediction.



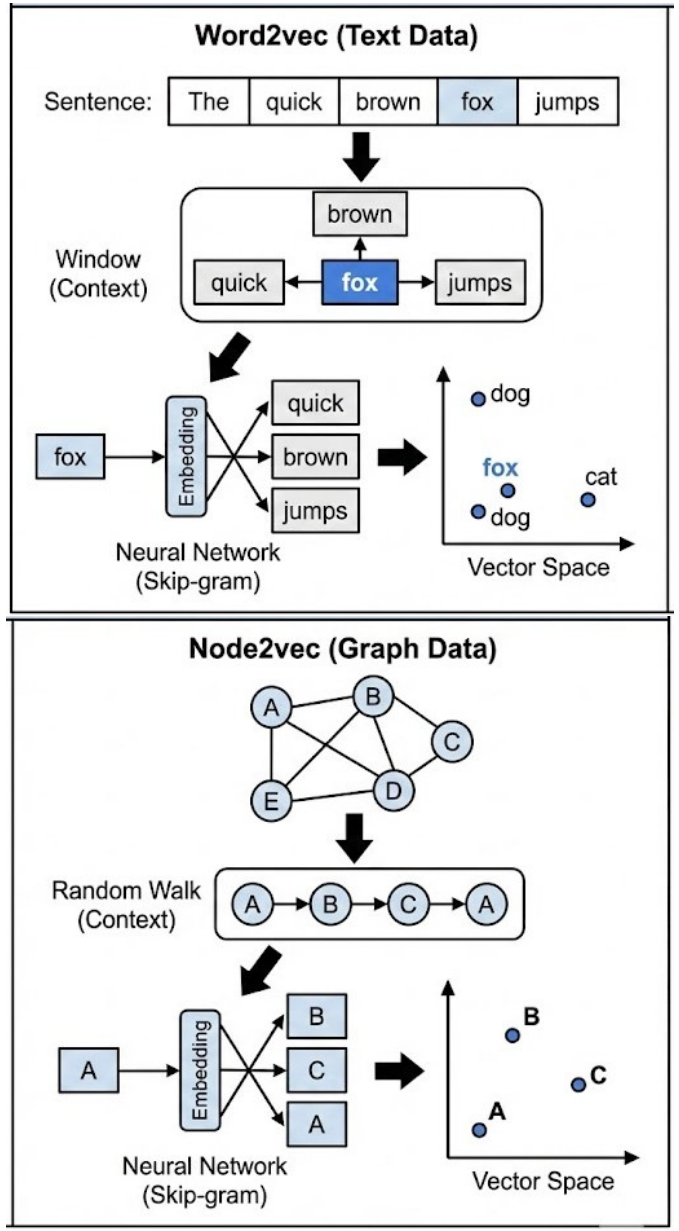
Limitations of Earlier Approaches

- Previous approaches to feature learning were *not expressive* enough to capture the diversity of connectivity patterns based on *structural equivalence* and *homophily* of nodes in a network.
- Hand engineered domain-specific features *cannot generalize* across different downstream tasks.
- Classical approaches based on *dimensionality reduction* like PCA are *expensive* as they involve eigen decomposition of relevant matrices.
- Other approaches inspired by the Word2Vec model provide no flexibility and rely on *rigid node sampling* strategies.

Analogy with Word2Vec

Word2Vec	Node2Vec
Words	Nodes
Sentence	Random Walk
Context Window	Node Neighbourhood
Document	Network

- As in Word2Vec, *similar words* should appear within *similar contexts*, we extend the analogy by assuming that *similar nodes* exist in *similar graph neighbourhoods*.
- We learn feature representations in a ***semi-supervised*** manner by optimizing the likelihood objective using *SGD* with *Negative Sampling*.
- Since network structures are not linear in nature like text, we *need a richer notion of neighbourhood* rather than the immediate neighbours.



The Feature Learning Framework

Our key contribution is in defining a flexible notion of a node's network neighborhood. By choosing an appropriate notion of a neighborhood, *node2vec* can learn representations that organize

- For any (un-)directed and (un-)weighted network $\boldsymbol{G(V, E)}$,
- $\forall \boldsymbol{u} \in \boldsymbol{V}$, let $\boldsymbol{N_S(u)} \subset \boldsymbol{V}$ be a network neighbourhood of $\boldsymbol{u}$ consisting of $\boldsymbol{k}$ nodes generated by a sampling strategy $\boldsymbol{S}$.
- Using the *Skip-Gram modelling approach*, we want to maximise the *log-probability* of observing neighbourhood $\boldsymbol{N\_S(u)}$ for node $\boldsymbol{u}$ conditioned on its feature representation $\boldsymbol{f(u)}$:

$$\max_f \sum_{u \in V} \log Pr(N_S(u)|f(u))$$

- Assuming conditional independence and symmetry of neighbouring nodes in feature space we have:

$$Pr(N_S(u)|f(u)) = \prod_{n_i \in N_S(u)} Pr(n_i|f(u)) \quad \Rightarrow \quad \max_f \sum_{u \in V} \left[-\log Z_u + \sum_{n_i \in N_S(u)} f(n_i) \cdot f(u) \right]$$

$$Pr(n_i|f(u)) = \frac{\exp(f(n_i) \cdot f(u))}{\sum_{v \in V} \exp(f(v) \cdot f(u))}$$

$Z_u = \sum_{v \in V} \exp(f(u) \cdot f(v))$ is the per node partition function approximated by using negative sampling.

The Sampling Strategy S

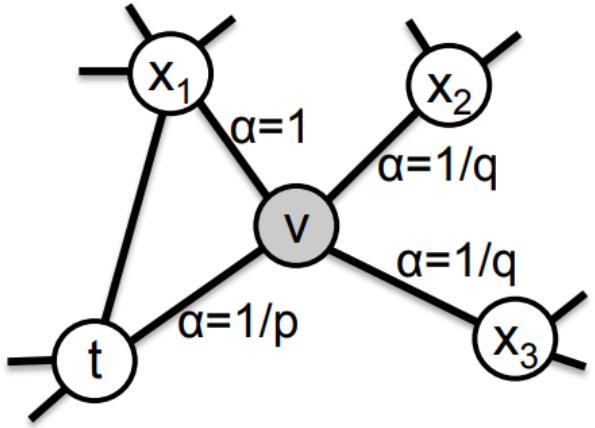


Figure 2: Illustration of the random walk procedure in *node2vec*. The walk just transitioned from t to v and is now evaluating its next step out of node v. Edge labels indicate search biases α.

$\boldsymbol{Z}$ is the normalization factor.

$\boldsymbol{\alpha_{pq}(t,x)}$ is the search bias for edge $\boldsymbol{w_{vx}}$.

Return param: $\boldsymbol{p}$ controls the likelihood of immediately revisiting a node in the walk.

In-Out param: $\boldsymbol{q}$ allows the search to differentiate between "inward" and "outward" nodes.

We define a *2nd order random walk* to generate the nodes $\boldsymbol{c_i}$ of a walk length $\boldsymbol{l}$ where $\boldsymbol{c_0} = \boldsymbol{u}$ with transition probability given by:

$$P(c_i = x \mid c_{i-1} = v) = \begin{cases} \frac{\pi_{vx}}{Z} & \text{if } (v,x) \in E \\ 0 & \text{otherwise} \end{cases}$$

We set the unnormalized probability of transitions as:

$$\pi_{vx} = \alpha_{pq}(t,x) \cdot w_{vx}$$

where

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

Smooth Interpolation of BFS and DFS

- If $\boldsymbol{p} > \boldsymbol{max}(\boldsymbol{q}, \mathbf{1})$ we are less likely to sample an already visited node in the next two steps encouraging moderate exploration and avoiding 2-hop redundancy in sampling.
- If $\boldsymbol{p} < \boldsymbol{min}(\boldsymbol{q}, \mathbf{1})$, it would lead the walk to backtrack a step and this would keep the walk close to the starting node u.
- If $\boldsymbol{q} > \mathbf{1}$, the random walk is likely to stay close to node t. This approximates a BFS.
- If $\boldsymbol{q} < \mathbf{1}$, the walk is more likely to visit nodes further away from the node t. This approximates a DFS.
- BFS obtains a microscopic view of the neighbourhood of every node which helps capture the structural equivalence of nodes.
- DFS obtains a macroscopic view of the neighbourhood, capturing the community structure based on the homophily of the nodes.
- Hence, the parameters $\boldsymbol{p}$ and $\boldsymbol{q}$ help the procedure to interpolate between BFS and DFS to capture different notions of node equivalences in a network.

The node2vec Algo

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** $walk$

- We simulate $\boldsymbol{r}$ random walks of length $\boldsymbol{l}$ for each node.
- The 3 stages of *preprocessing the transition probabilities*, *simulating the random walks* and *SGD* are parallelizable and run sequentially.
- The space required for precomputing transition probabilities is $\mathbf{O}(\boldsymbol{a^2}|\boldsymbol{V}|)$ where $\boldsymbol{a}$ is the average node degree of the network.
- Sampling of the probabilities is done using AliasSampling in $\mathbf{O}(\mathbf{1})$.
- By simulating a random walk of length $\boldsymbol{l}$, we can generate $\boldsymbol{k}$ samples for $(\boldsymbol{l} - \boldsymbol{k})$ nodes making the effective sampling time $\mathbf{O}\left(\frac{l}{k(l\text{-}k)}\right)$.

Learning Edge Representations

- We extend the representations learnt for nodes to pairs of nodes using a bootstrapping approach.
- Given nodes $\boldsymbol{u}$ and $\boldsymbol{v}$ we can define a binary operation $\boldsymbol{g}: \boldsymbol{V} \times \boldsymbol{V} \rightarrow \mathbb{R}^{\boldsymbol{d'}}$ on the embeddings $\boldsymbol{f}(\boldsymbol{u})$ and $\boldsymbol{f}(\boldsymbol{v})$ to create an embedding function $\boldsymbol{g}(\boldsymbol{u}, \boldsymbol{v})$ that gives us the edge representation for edge $(\boldsymbol{u}, \boldsymbol{v})$ where $\boldsymbol{d'}$ is the dimension of the edge embedding space.
- Some binary operations which preserve $\boldsymbol{d} = \boldsymbol{d'}$ are given below:

Operator	Symbol	Definition
Average	$\boxplus$	$[f(u) \boxplus f(v)]_i = \frac{f_i(u)+f_i(v)}{2}$
Hadamard	$\boxdot$	$[f(u) \boxdot f(v)]_i = f_i(u) * f_i(v)$
Weighted-L1	$\|\cdot\|_{\bar{1}}$	$\|f(u) \cdot f(v)\|_{\bar{1}i} = \|f_i(u) - f_i(v)\|$
Weighted-L2	$\|\cdot\|_{\bar{2}}$	$\|f(u) \cdot f(v)\|_{\bar{2}i} = \|f_i(u) - f_i(v)\|^2$

Experiments and Results

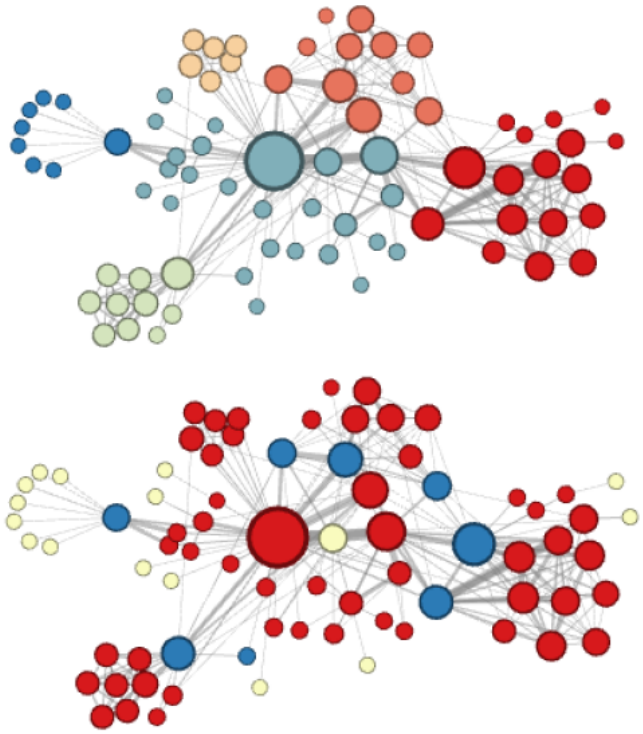


Figure 3: Complementary visualizations of Les Misérables coappearance network generated by *node2vec* with label colors reflecting homophily (top) and structural equivalence (bottom).

(Top) shows clustering when we set $\boldsymbol{p = 1, q = 0.5}$.
(Bottom) shows the same when we set $\boldsymbol{p = 1, q = 2}$.

Algorithm	**Dataset**		
	BlogCatalog	PPI	Wikipedia
Spectral Clustering	0.0405	0.0681	0.0395
DeepWalk	0.2110	0.1768	0.1274
LINE	0.0784	0.1447	0.1164
node2vec	**0.2581**	**0.1791**	**0.1552**
node2vec settings (p,q)	0.25, 0.25	4, 1	4, 0.5
Gain of *node2vec* [%]	**22.3**	**1.3**	**21.8**

Table 2: Macro-F_1 scores for multilabel classification on BlogCatalog, PPI (Homo sapiens) and Wikipedia word cooccurrence networks with 50% of the nodes labeled for training.

Op	**Algorithm**	**Dataset**		
		Facebook	PPI	arXiv
	Common Neighbors	0.8100	0.7142	0.8153
	Jaccard's Coefficient	0.8880	0.7018	0.8067
	Adamic-Adar	0.8289	0.7126	0.8315
	Pref. Attachment	0.7137	0.6670	0.6996
(a)	Spectral Clustering	0.5960	0.6588	0.5812
	DeepWalk	0.7238	0.6923	0.7066
	LINE	0.7029	0.6330	0.6516
	node2vec	0.7266	0.7543	0.7221
(b)	Spectral Clustering	0.6192	0.4920	0.5740
	DeepWalk	**0.9680**	0.7441	0.9340
	LINE	0.9490	0.7249	0.8902
	node2vec	**0.9680**	**0.7719**	**0.9366**
(c)	Spectral Clustering	0.7200	0.6356	0.7099
	DeepWalk	0.9574	0.6026	0.8282
	LINE	0.9483	0.7024	0.8809
	node2vec	0.9602	0.6292	0.8468
(d)	Spectral Clustering	0.7107	0.6026	0.6765
	DeepWalk	0.9584	0.6118	0.8305
	LINE	0.9460	0.7106	0.8862
	node2vec	0.9606	0.6236	0.8477

Table 4: Area Under Curve (AUC) scores for link prediction. Comparison with popular baselines and embedding based methods bootstapped using binary operators: (a) Average, (b) Hadamard, (c) Weighted-L1, and (d) Weighted-L2 (See Table 1 for definitions).

Experiments and Results

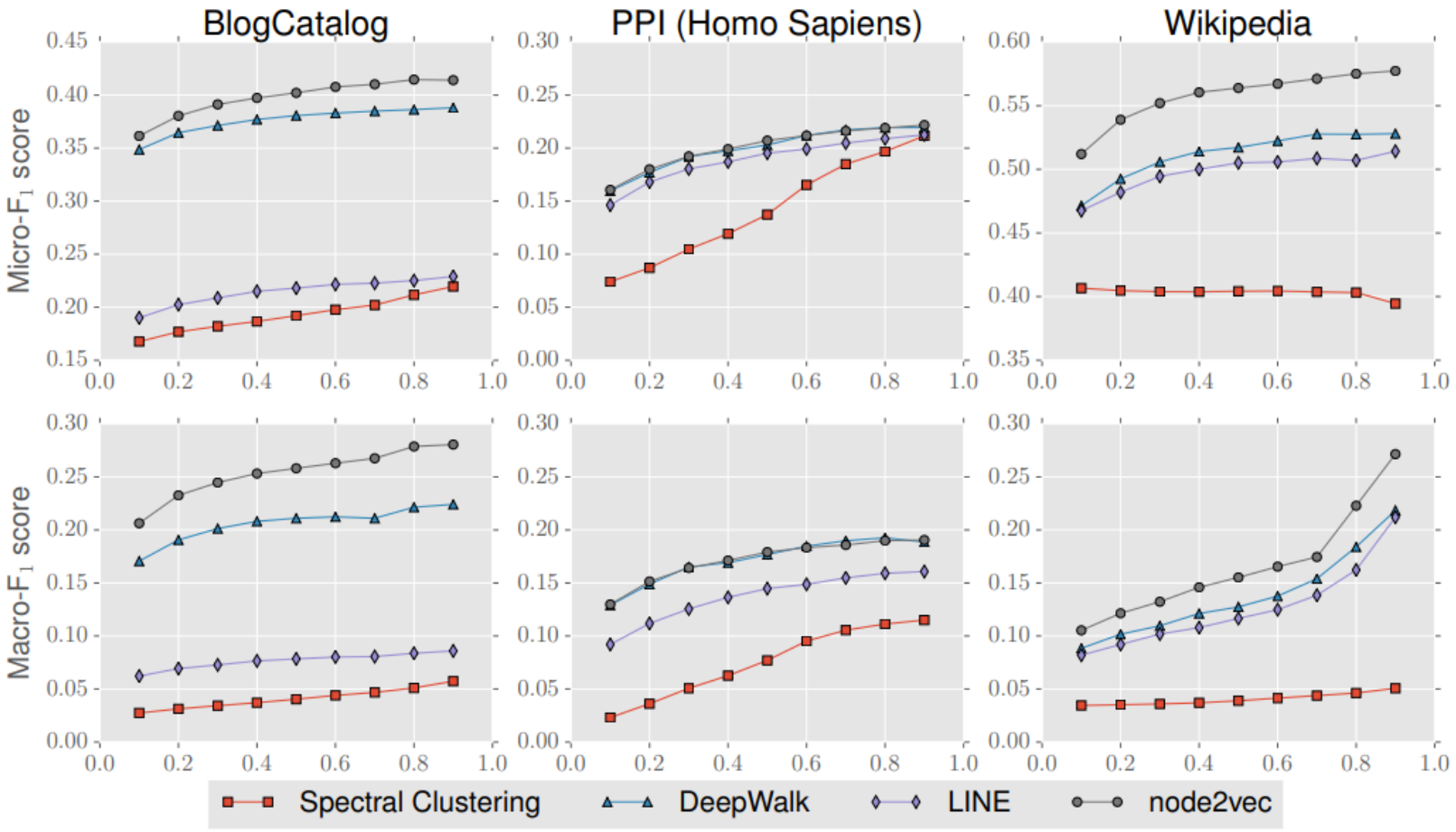


Figure 4: Performance evaluation of different benchmarks on varying the amount of labeled data used for training. The x axis denotes the fraction of labeled data, whereas the y axis in the top and bottom rows denote the Micro-F_1 and Macro-F_1 scores respectively. DeepWalk and *node2vec* give comparable performance on PPI. In all other networks, across all fractions of labeled data *node2vec* performs best.

Parameter Sensitivity, Perturbations, Scalability

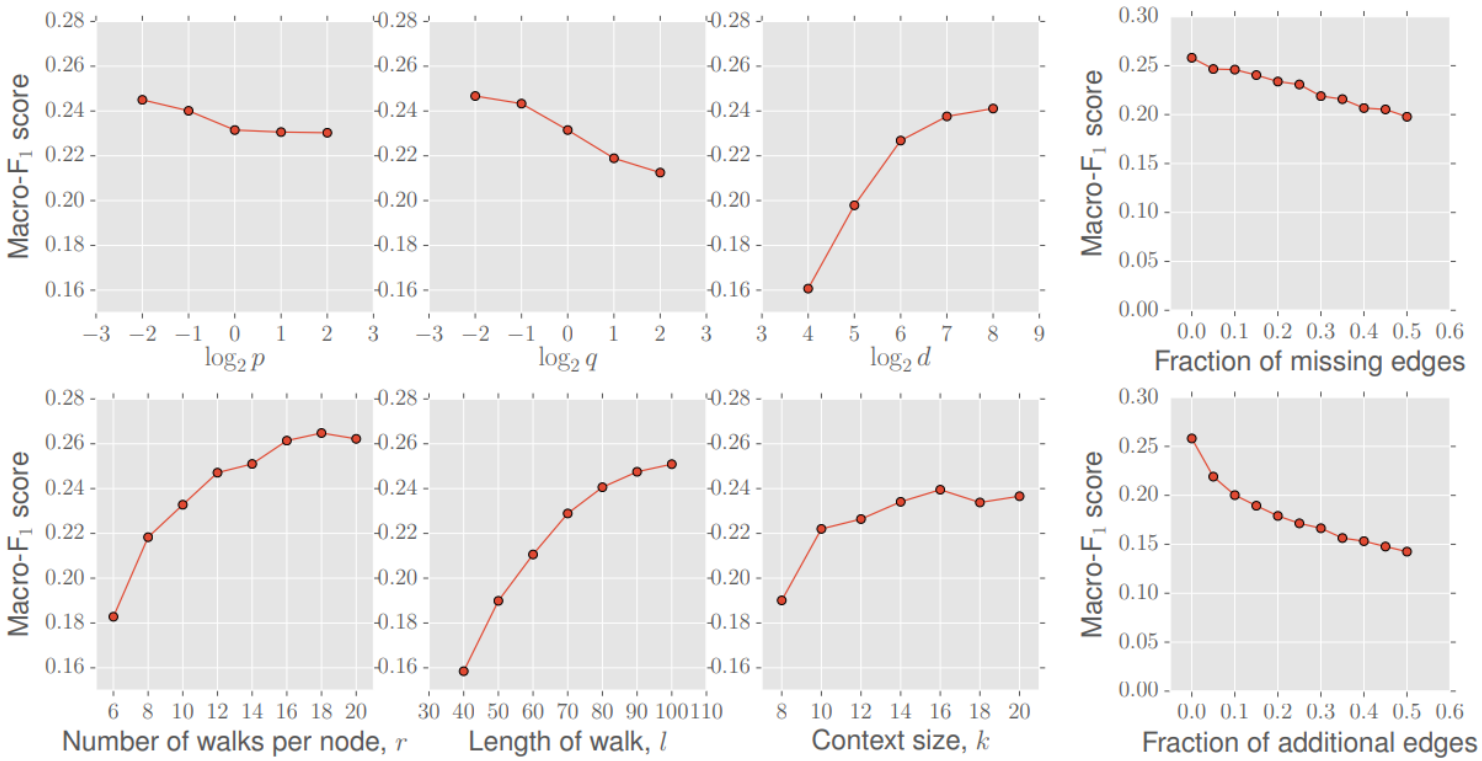


Figure 5: (a). Parameter sensitivity (b). Perturbation analysis for multilabel classification on the BlogCatalog network.

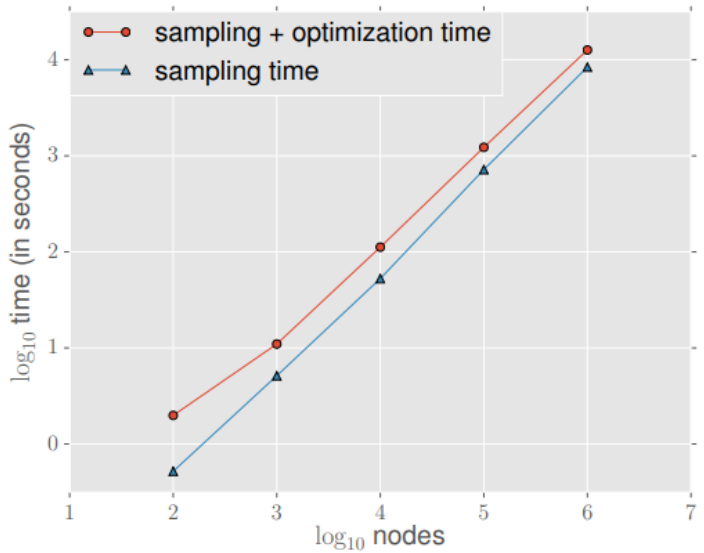


Figure 6: Scalability of *node2vec* on Erdos-Renyi graphs with an average degree of 10.

One interesting thing to do here could be to plot performance as a function of the ratio $^{p}/_{q}$

Strengths & Weaknesses

Strengths

- Automated feature extraction based on graph topology
- Flexibility with controllable parameters like $\boldsymbol{p}$ and $\boldsymbol{q}$ to capture homophily and structural equivalence
- Parallelizable with efficient sampling mechanism
- Scalable and robust
- Strong downstream task performance

Weaknesses

- Cannot take into account node features if available (GNNs can)
- Cannot handle heterogenous graphs (extensions like Het-node2vec exist)
- Cannot embed new nodes (OOV problem)
- Cannot handle dynamic graphs (versions like node2vec-online exist)

Things to do next:

- Study the performance of node2vec as a function of the ratio p/q.
- Study node2vec extensions for heterogenous graphs and dynamic graphs.
- Study GNNs and attention mechanisms in GNNs.

References

1. Grover, A. and Leskovec, J. 2016. *node2vec: Scalable Feature Learning for Networks*. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (KDD '16). Association for Computing Machinery, New York, NY, USA, 855–864. DOI: https://doi.org/10.1145/2939672.2939754.
2. Tomaz Bratanic: Complete guide to understanding Node2Vec algorithm on Medium. (https://medium.com/data-science/complete-guide-to-understanding-node2vec-algorithm-4e9a35e5d147)
3. OpenAI. (2025). ChatGPT (Dec 25 version) [Large language model]. (https://chat.openai.com/chat).
4. Google DeepMind. (2025). Nano Banana Pro (Dec 25 version) [Image generation model] (https://gemini.google.com/)

THANKS!

Thanks PreCog @ IIIT-Hyd for providing this opportunity.

I learnt a lot and had fun!