node2vec

Scalable Feature Learning for Networks

Seunghan Lee (CSE-URP)

20.02.17(Mon)

Goal

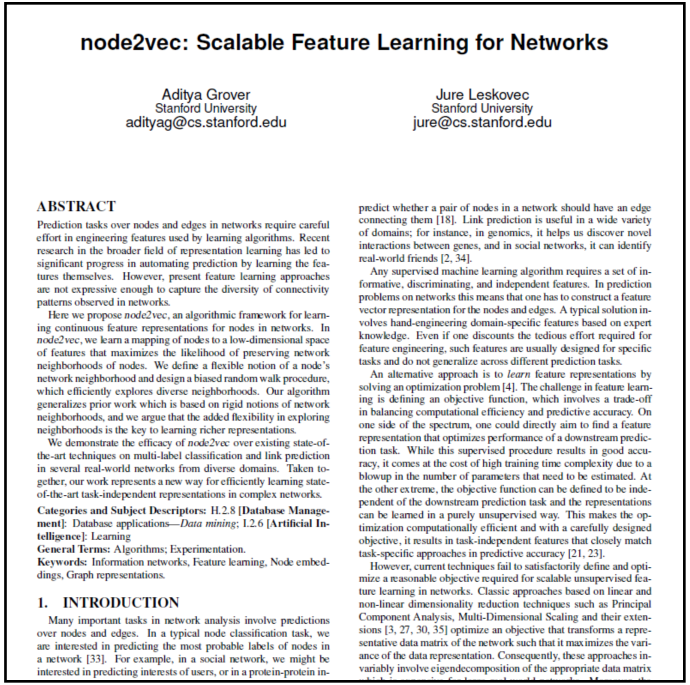
Summary of node2vec

1. node2vec algorithm

- about DFS & BFS

2. node2vec implementation

- with numpy



node2vec: Scalable Feature Learning for Networks

Aditya Grover
Stanford University
adityag@cs.stanford.edu

Jure Leskovec
Stanford University
jure@cs.stanford.edu

ABSTRACT

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.

Categories and Subject Descriptors: H.2.8 [**Database Management**]: Database applications—*Data mining*; I.2.6 [**Artificial Intelligence**]: Learning
General Terms: Algorithms; Experimentation.
Keywords: Information networks, Feature learning, Node embeddings, Graph representations.

1. INTRODUCTION

Many important tasks in network analysis involve predictions over nodes and edges. In a typical node classification task, we are interested in predicting the most probable labels of nodes in a network [33]. For example, in a social network, we might be interested in predicting interests of users, or in a protein-protein in-

predict whether a pair of nodes in a network should have an edge connecting them [18]. Link prediction is useful in a wide variety of domains; for instance, in genomics, it helps us discover novel interactions between genes, and in social networks, it can identify real-world friends [2, 34].

Any supervised machine learning algorithm requires a set of informative, discriminating, and independent features. In prediction problems on networks this means that one has to construct a feature vector representation for the nodes and edges. A typical solution involves hand-engineering domain-specific features based on expert knowledge. Even if one discounts the tedious effort required for feature engineering, such features are usually designed for specific tasks and do not generalize across different prediction tasks.

An alternative approach is to *learn* feature representations by solving an optimization problem [4]. The challenge in feature learning is defining an objective function, which involves a trade-off in balancing computational efficiency and predictive accuracy. On one side of the spectrum, one could directly aim to find a feature representation that optimizes performance of a downstream prediction task. While this supervised procedure results in good accuracy, it comes at the cost of high training time complexity due to a blowup in the number of parameters that need to be estimated. At the other extreme, the objective function can be defined to be independent of the downstream prediction task and the representations can be learned in a purely unsupervised way. This makes the optimization computationally efficient and with a carefully designed objective, it results in task-independent features that closely match task-specific approaches in predictive accuracy [21, 23].

However, current techniques fail to satisfactorily define and optimize a reasonable objective required for scalable unsupervised feature learning in networks. Classic approaches based on linear and non-linear dimensionality reduction techniques such as Principal Component Analysis, Multi-Dimensional Scaling and their extensions [3, 27, 30, 35] optimize an objective that transforms a representative data matrix of the network such that it maximizes the variance of the data representation. Consequently, these approaches invariably involve eigendecomposition of the appropriate data matrix

Contents

1. Introduction

Contribution of node2vec

1. Introduction

contribution of node2vec

1. Efficient scalable algorithm for feature learning in networks (using SGD)

2. Provides flexibility in discovering representations

3. Extend node2vec from ‘nodes’ to ‘edges’

4. Evaluate node2vec for (1) multi-label classification & (2) link prediction

1. Introduction

contribution of node2vec

1. Efficient scalable algorithm for feature learning in networks (using SGD)

(SGD with “negative sampling” -> efficient in huge network)

2. Provides flexibility in discovering representations

3. Extend node2vec from ‘nodes’ to ‘edges’

4. Evaluate node2vec for (1) multi-label classification & (2) link prediction

1. Introduction

contribution of node2vec

1. Efficient scalable algorithm for feature learning in networks (using SGD)

   (SGD with "negative sampling" -> efficient in huge network)

2. Provides flexibility in discovering representations

   (provides parameter to tune the explored search space -> BFS, DFS)

3. Extend node2vec from 'nodes' to 'edges'

4. Evaluate node2vec for (1) multi-label classification & (2) link prediction

1. Introduction

contribution of node2vec

1. Efficient scalable algorithm for feature learning in networks (using SGD)

(SGD with “negative sampling” -> efficient in huge network)

2. Provides flexibility in discovering representations

(provides parameter to tune the explored search space -> BFS, DFS)

3. Extend node2vec from ‘nodes’ to ‘edges’

(classic : node embedding -> node2vec : node embedding & edge embedding)

4. Evaluate node2vec for (1) multi-label classification & (2) link prediction

1. Introduction

contribution of node2vec

1. Efficient scalable algorithm for feature learning in networks (using SGD)

   (SGD with "negative sampling" -> efficient in huge network)

2. Provides flexibility in discovering representations

   (provides parameter to tune the explored search space -> BFS, DFS)

3. Extend node2vec from 'nodes' to 'edges'

   (classic : node embedding -> node2vec : node embedding & edge embedding)

4. Evaluate node2vec for (1) multi-label classification & (2) link prediction

   ((1) multi-label classification : classify which class the node belongs to)

   ((2) link prediction : predict if there is a link between to nodes)

1. Introduction

contribution of node2vec

1. Efficient scalable algorithm for feature learning in networks (using SGD)

(SGD with "negative sampling" -> efficient in huge network)

2. Provides flexibility in discovering representations

(provides parameter to tune the explored search space -> **BFS, DFS**)

3. Extend node2vec from 'nodes' to 'edges'

(classic : node embedding -> node2vec : node embedding & **edge embedding**)

4. Evaluate node2vec for (1) multi-label classification & (2) link prediction

((1) multi-label classification : classify which class the node belongs to)

((2) **link prediction** : predict if there is a link between to nodes)

2. node2vec algorithm

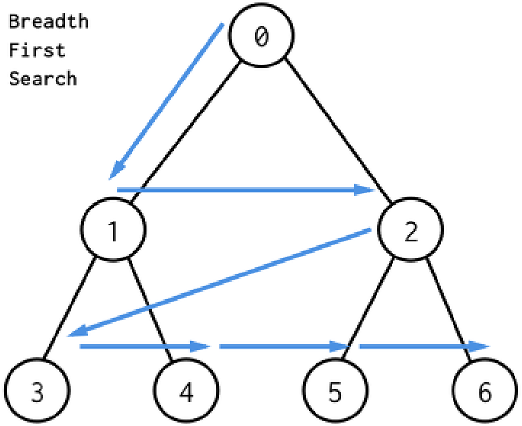
1) Classic Search Algorithm : BFS & DFS
2) Random Walk in node2vec

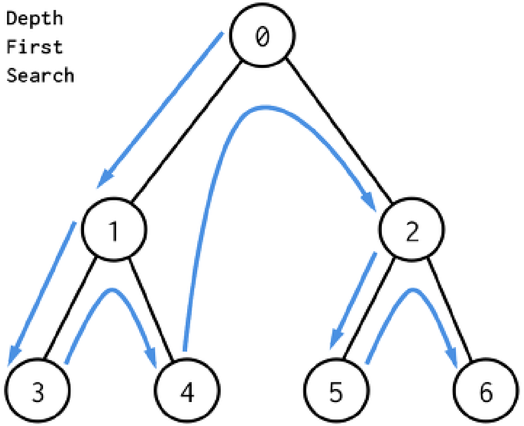
2. node2vec algorithm

1. Classic Search Algorithm : BFS & DFS

“An algorithm for traversing or searching tree or graph data structures” **(wikipedia)**

https://t1.daumcdn.net/cfile/tistory/997183445C7625B921





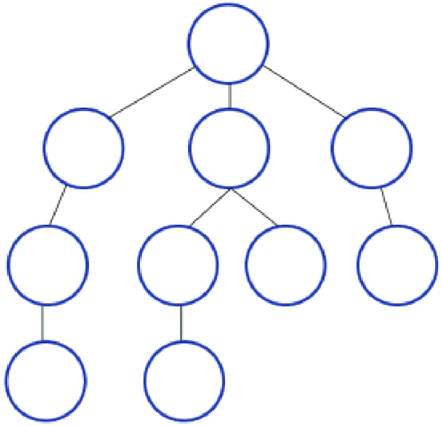
BFS (Breadth-First Search)

DFS (Depth-First Search)

2. node2vec algorithm

1. Classic Search Algorithm : BFS & DFS

BFS (Breadth-First Search)



https://seing.tistory.com/29

Search the node with the same level (breadth)!

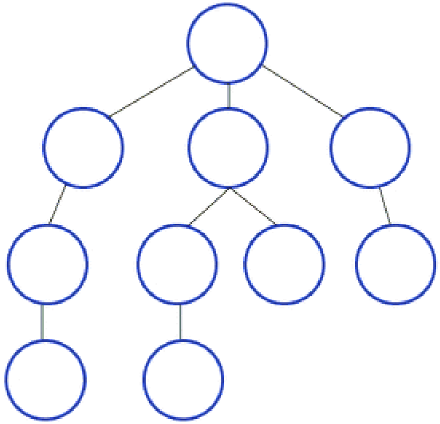
After finishing searching the certain level (breadth),

get down to the below level (depth)!

2. node2vec algorithm

1. Classic Search Algorithm : BFS & DFS

DFS (Depth-First Search)



https://seing.tistory.com/29

Search the child node first!

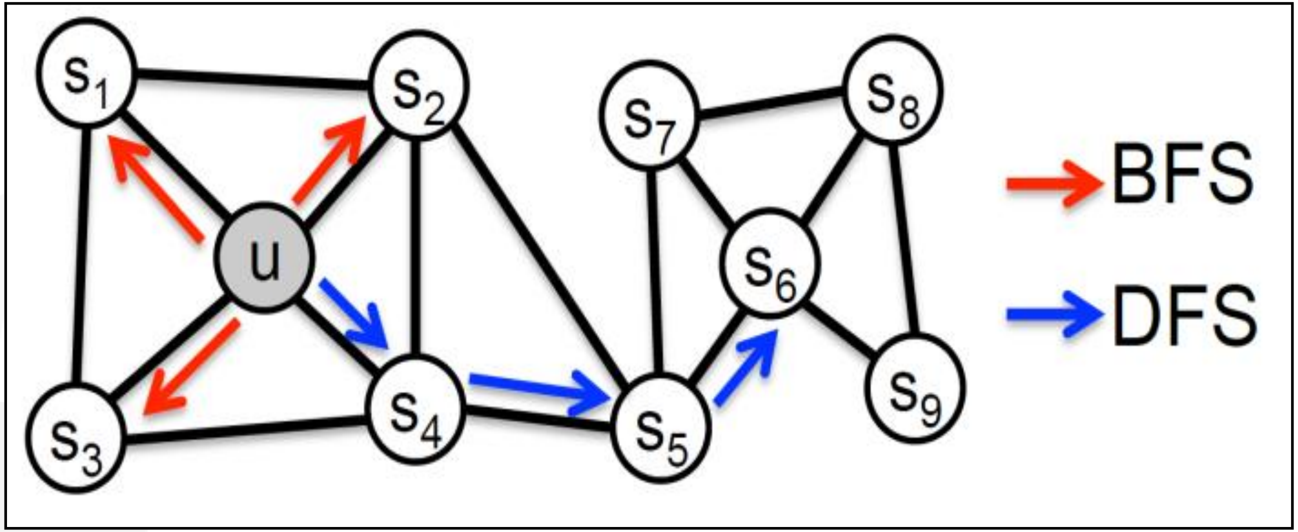
After finishing searching the last child node,

“Backtracking” (returning back to the parent node)

2. node2vec algorithm

1. Classic Search Algorithm : BFS & DFS

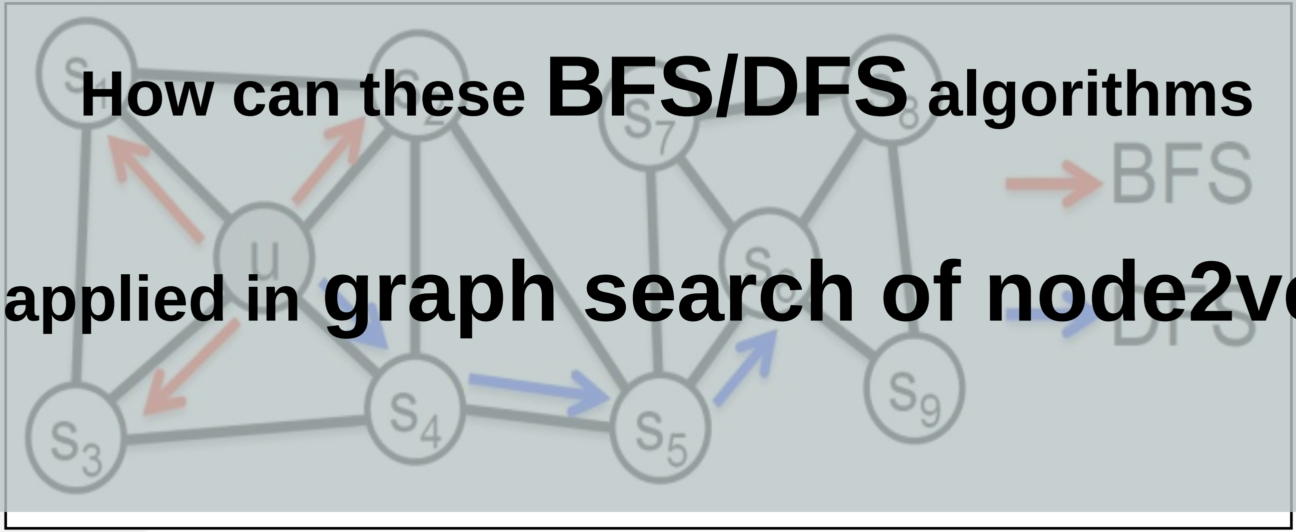
BFS & DFS in graph



2. node2vec algorithm

1. Classic Search Algorithm : BFS & DFS

BFS & DFS in graph



How can these BFS/DFS algorithms be applied in graph search of node2vec?

2. node2vec algorithm

1. Classic Search Algorithm : BFS & DFS

before moving on…

In the phrase “***Nodes with high similarity should be also embedded closely in the representative space***”,

how can we define the word **“similarity”** ?

2. node2vec algorithm

1. Classic Search Algorithm : BFS & DFS

before moving on…

In the phrase "***Nodes with high similarity should be also embedded closely in the representative space***",

how can we define the word **"similarity"** ?

"**how to sample the random walk**" is affected by "**how we define 'similarity'** "

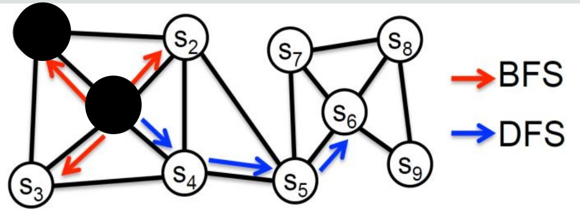
2. node2vec algorithm

1. Classic Search Algorithm : BFS & DFS

Two kinds of Similarity

1. Homophily

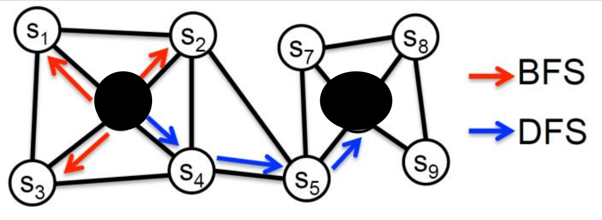
nodes that are "**highly interconnected**" and belong to similar network clusters should be embedded closely together



Node2vec_Scalable Feature Learning for Networks

2. Structural Equivalence

nodes that have "**similar structural roles**" in networks should be embedded closely together



Node2vec_Scalable Feature Learning for Networks

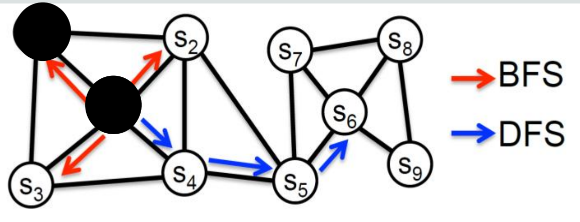
2. node2vec algorithm

1. Classic Search Algorithm : BFS & DFS

Two kinds of Similarity

1. Homophily

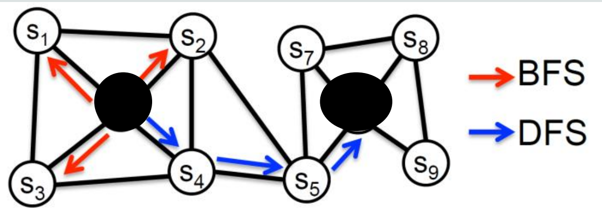
Networks that are “**highly interconnected**” and belong to similar network clusters should be embedded closely together



Node2vec_Scalable Feature Learning for Networks

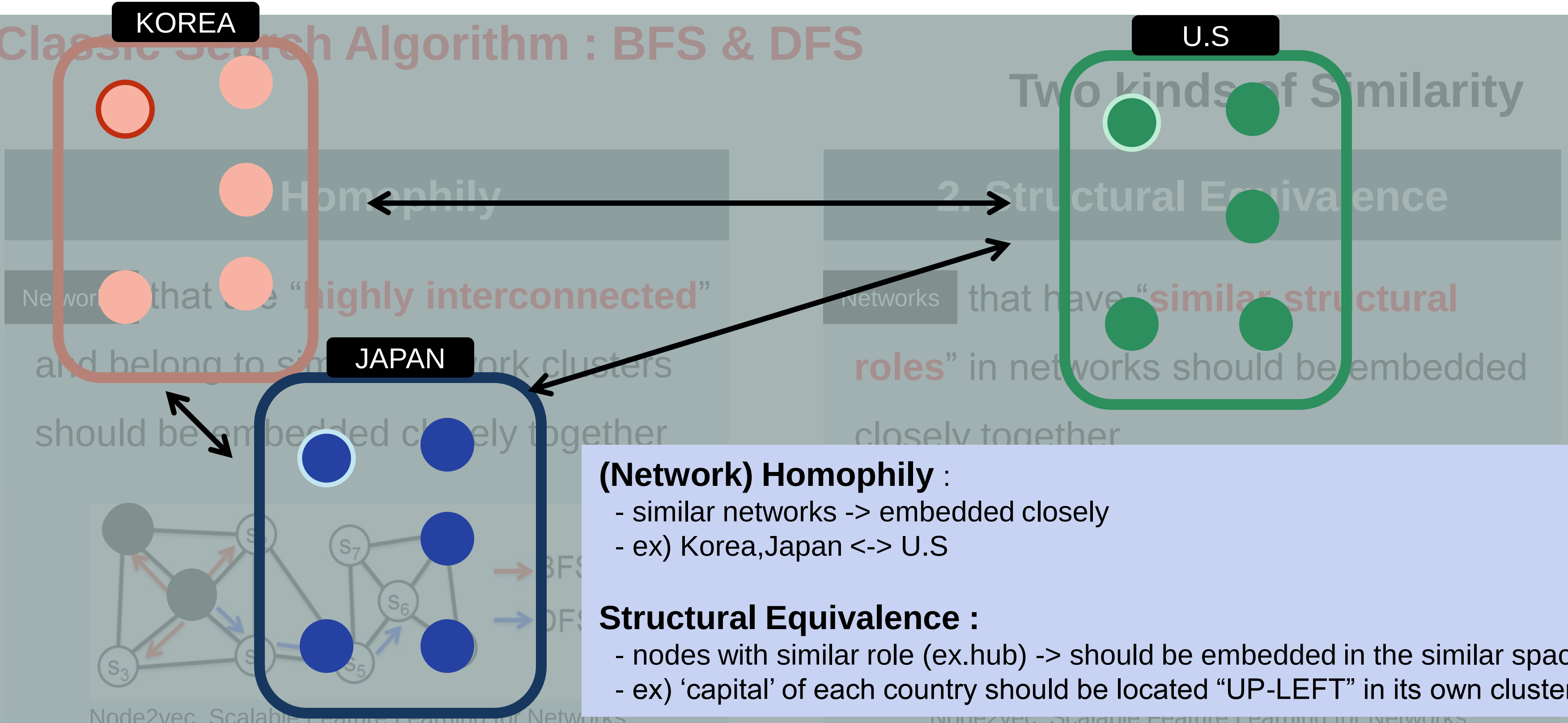
2. Structural Equivalence

nodes that have “**similar structural roles**” in networks should be embedded in the similar space inside its own network



Node2vec_Scalable Feature Learning for Networks

2. node2vec algorithm



(Network) Homophily :
- similar networks -> embedded closely
- ex) Korea,Japan <-> U.S

Structural Equivalence :
- nodes with similar role (ex.hub) -> should be embedded in the similar space
- ex) 'capital' of each country should be located "UP-LEFT" in its own cluster

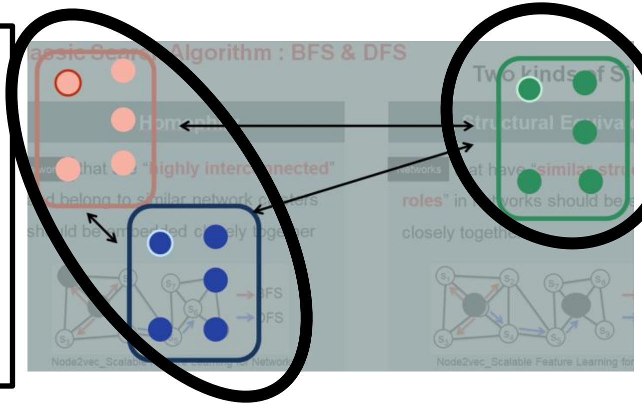
2. node2vec algorithm

1. Classic Search Algorithm : BFS & DFS

BFS & DFS in node2vec

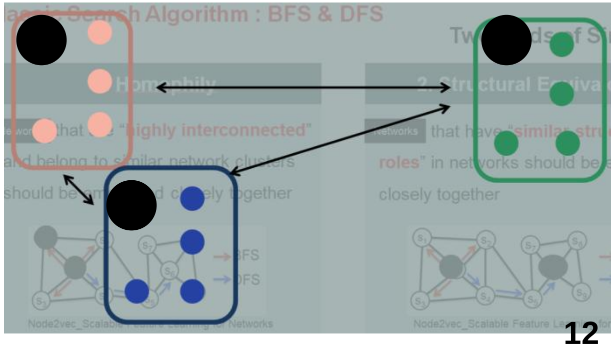
DFS (Depth-first Sampling)

- neighborhood : nodes sequentially sampled at increasing distance from the source node
- macro view
- to capture “homophily”



BFS (Breadth-first Sampling)

- neighborhood : only immediate neighbors of the source node
- micro view
- to capture “structural equivalence”



2. node2vec algorithm

2. Random Walk in node2vec

Considers both "**homophily**" and "**structural equivalence**"

(with DFS) (with BFS)

- These two similarities are not exclusive!

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

- Introduce a **"search bias"** term

(Define a "**second order random walk**" with two parameters p & q)

2. node2vec algorithm

2. Random Walk in node2vec

Transition probability

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

Search bias α Weight of edge

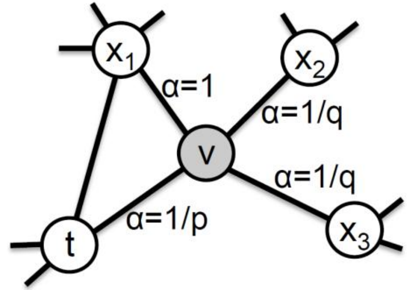
normalize

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

2. node2vec algorithm

2. Random Walk in node2vec

Search bias α



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

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

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

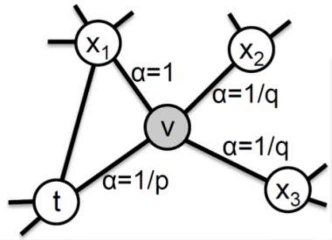
t : previous node / **v** : current node / **x** : node to choose as the next step

Consider the previous node(t) to sample the next node!

2. node2vec algorithm

2. Random Walk in node2vec

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



return parameter, p

If p > max(q,1) :

getting further from the previous node! DFS

If p < min(q,1) :

getting closer to the previous node (Local search) BFS

return parameter, q

If q > 1:

getting closer to the previous node! BFS

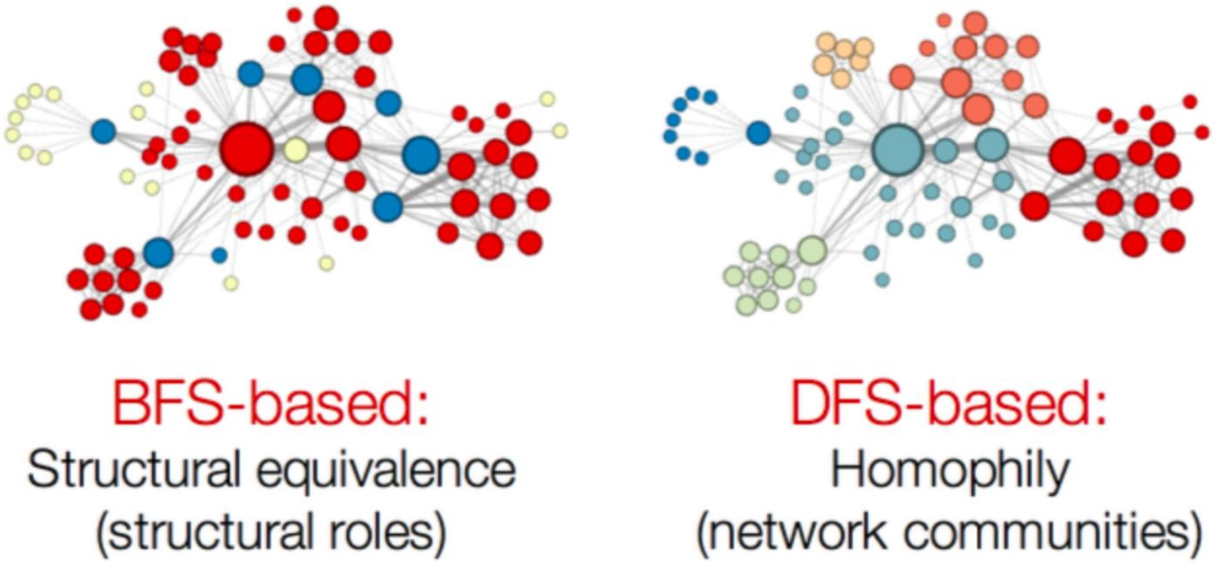
If q < 1:

getting further from the previous node! DFS

2. node2vec algorithm

2. Random Walk in node2vec

http://nocotan.github.io/images/20170701/fig2.png



Tune p & q to choose the similarity to focus on!

2. node2vec algorithm

2. Random Walk in node2vec

Benefits of Random Walk

참고 : Node2vec(서창원)

Space complexity $O(a^2\|V\|)$	Time complexity $O(\frac{l}{k(l-k)})$

Space complexity to store immediate neighbors : $O(|E|)$

$$2|E| = \sum_{v \in V} deg(v)$$

$$= \frac{\sum_{v \in V} deg(v)}{|V|}|V|$$

$$= a|V| \ where \ a \ is \ the \ average \ degree \ of \ V$$

$$random \ walk = \{u, s_4, s_5, s_6, s_7, s_8, s_9\} \quad l = 7 \quad k = 5$$

$$N_S(u) = \{s_4, s_5, s_6, s_7, s_8\}$$

$$N_S(s_4) = \{s_5, s_6, s_7, s_8, s_9\}$$

3. node2vec implementation

1) Embedding
2) Classification with MLP & Logistic Regression

3. node2vec implementation

1. Embedding

1

1) sigmoid

```
def sigmoid(x):
    return 1/(1+np.exp(-x))
```

Sigmoid function

2

2) pos_list & neg_list : getting the positive & negative nodes

```
def pos_list(node):
    return np.nonzero(A[node])[1]

def neg_list(node):
    return np.where(A[node]==0)[1]
```

Getting the positive & negative nodes

3

3) next_choice

- (1) previous : 't'
- (2) now : 'v'
- (3) next : 'x'

```
def next_choice(v,t,p,q):
    positive = pos_list(v)
    li = np.array([])
    for pos in positive:
        if pos==t:
            li = np.append(li,1/p)
        elif pos in pos_list(t):
            li = np.append(li,1)
        else :
            li = np.append(li,1/q)

    prob = li/li.sum()

    return np.random.choice(positive,1,p=prob)[0]
```

Choosing the next step, based on transition probability (considering search bias)

3. node2vec implementation

1. Embedding

4

4) random_step : getting the random step, using next_choice

```
def random_step(v,num_walk,p,q):
    t = np.random.choice(pos_list(v)) # (1) previous

    walk_list = [v]
    for _ in range(num_walk):
        x = next_choice(v,t,p,q)
        walk_list.append(x)
        v = x
        t = v
    return walk_list
```

Random Step :

make the random step of length ‘num_walk’,

based on ‘next_choice’ (considering BFS &DFS)

3. node2vec implementation

1. Embedding

4

4) random_step : getting the random step, using next_choice

```python
def random_step(v,num_walk,p,q):
    t = np.random.choice(pos_list(v)) # (1) previous

    walk_list = [v]
    for _ in range(num_walk):
        x = next_choice(v,t,p,q)
        walk_list.append(x)
        v = x
        t = v
    return walk_list
```

Random Step :

make the random step of length ‘num_walk’,

based on ‘next_choice’ (considering BFS &DFS)

1	sigmoid
2	pos_list & neg_list
3	next_choice
4	random_step

⬇

node2vec

3. node2vec implementation

1. Embedding

3. node2vec

```
def node2vec(dim,num_epoch,length,lr,k,p,q,num_neg):
    embed = np.random.random((A.shape[0],dim))

    for epoch in range(num_epoch):
        for v in np.arange(A.shape[0]):
            walk = random_step(v,length-1,p,q)  # (1) random walk

            for idx in range(length-k):

                not_neg_list = np.append(walk[max(0,idx-k):idx+k],pos_list(walk[idx]))
                neg_list = list(set(np.arange(A.shape[0])) - set(not_neg_list))
                random_neg = np.random.choice(neg_list,num_neg,replace=False)

                for pos in range(idx+1,idx+k+1):
                    if walk[idx]!=walk[pos]:
                        pos_embed = embed[walk[pos]]
                        embed[walk[idx]] -= lr * (sigmoid(np.dot(embed[walk[idx]],pos_embed))-1) * pos_embed

                for neg in random_neg:
                    neg_embed = embed[neg]
                    embed[walk[idx]] -= lr * (sigmoid(np.dot(embed[walk[idx]],neg_embed))) * neg_embed

    return embed
```

1. Take random step (with length ‘length’)

2. Negative sampling (with size ‘num_neg’)

3. Update with “positive” samples

4. Update with “negative” samples

input

- dim (dimension to reduce)
- num_epoch (number of epoch)
- length (walk length)
- lr (learning rate)
- k (context size a)
- p & q (parameter for search bias)
- num_neg (number of negative samples)

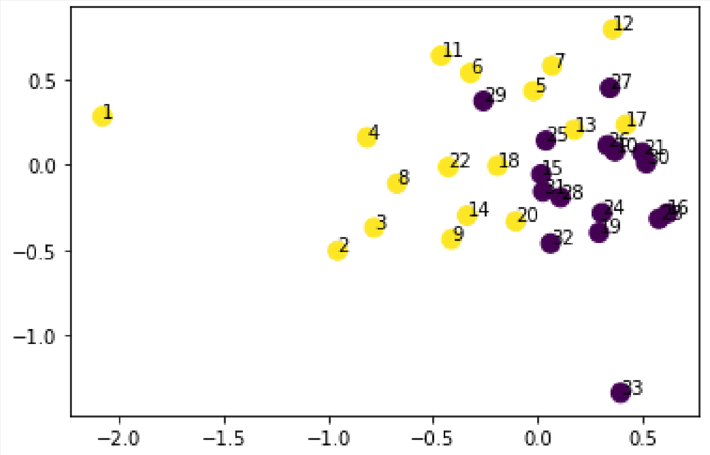
output

Embedded Vector

3. node2vec implementation

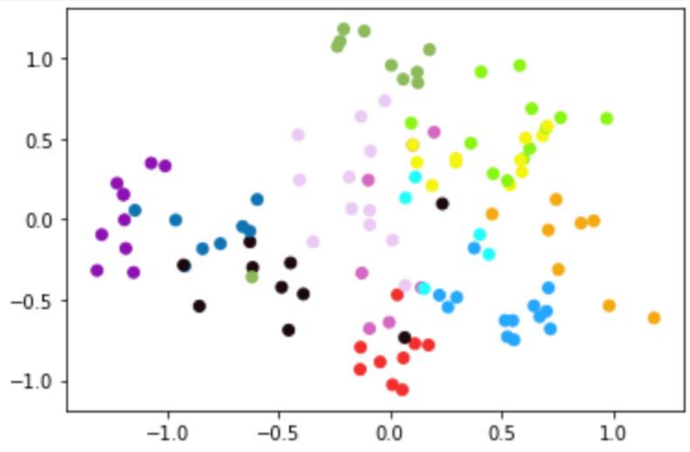
1. Embedding

Result : Embedded into 2-dimension




```
embed = node2vec(dim=2,num_epoch=10,length=8,lr=0.02,
k=2,p=2,q=2,num_neg=5)
```

1. Karate




```
embed = node2vec(dim=2,num_epoch=50,length=8,lr=0.02,
k=2,p=2,q=2,num_neg=5)
```

2. Football

3. node2vec implementation

2. Classification

1) Karate (Logistic Regression & MLP)

2) Football (OVR & MLP)

(+ comparison with other methods)

(참고 : Node2vec_project(김주현))

3. node2vec implementation

2. Classification

1) Karate (Logistic Regression & MLP)

	10%	30%	50%	70%
DeepWalk(p=1, q=1)	78.95%	96.0%	100.0%	100.0%
p=1, q=0.5	50.0%	100.0%	94.12%	100.0%
p=1, q=2	93.75%	92.31%	100.0%	100.0%

Logistic Regression

(epoch = 800, lr = 0.05)

	10%	30%	50%	70%
DeepWalk(p=1, q=1)	70.59%	82.76%	93.33%	88.89%
p=1, q=0.5	68.09%	76.92%	100.0%	100.0%
p=1, q=2	90.91%	96.0%	100.0%	100.0%

Multi Layer Perceptron

(epoch = 1000, lr = 0.001)

3. node2vec implementation

2. Classification

2) Football (OVR & MLP)

(1) OVR

DeepWalk

	10%	30%	50%	70%
Macro F1-score	66.63%	65.43%	77.35%	82.13%
Micro F1-score	72.82%	69.14%	82.46%	88.57%

Line with First Order Proximity

	10%	30%	50%	70%
Macro F1-score	45.62%	77.99%	81.3%	78.28%
Micro F1-score	50.49%	83.95%	87.72%	88.57%

Line with Second Order Proximity

	10%	30%	50%	70%
Macro F1-score	23.95%	53.79%	50.7%	58.21%
Micro F1-score	29.13%	56.79%	49.12%	62.86%

Node2Vec

	10%	30%	50%	70%
Macro F1-score	48.57%	64.01%	80.12%	93.17%
Micro F1-score	58.25%	66.67%	85.96%	94.29%

3. node2vec implementation

2. Classification

2) Football (OVR & MLP)

(2) MLP

DeepWalk

	10%	30%	50%	70%
Macro F1-score	12.72%	7.59%	3.69%	65.32%
Micro F1-score	19.42%	17.28%	12.28%	74.29%

Line with First Order Proximity

	10%	30%	50%	70%
Macro F1-score	4.61%	48.83%	6.36%	18.67%
Micro F1-score	12.62%	54.32%	14.04%	26.32%

Line with Second Order Proximity

	10%	30%	50%	70%
Macro F1-score	1.34%	1.36%	0.6%	1.32%
Micro F1-score	8.74%	8.64%	3.51%	8.57%

Node2Vec

	10%	30%	50%	70%
Macro F1-score	7.02%	17.67%	55.0%	80.32%
Micro F1-score	16.5%	33.33%	57.89%	88.57%

Reference

[1] Aditya Grover : *node2vec : Scalable Feature Learning for Networks*

[2] 서창원 : *node2vec*

[3] 김주현 : *node2vec project*

Thank You!!