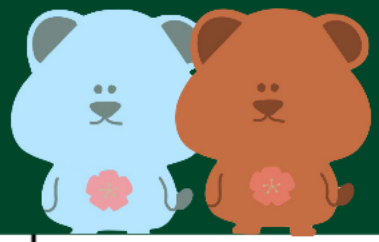


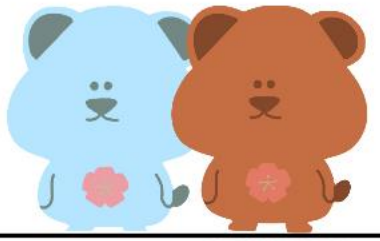
2주차 세션

김나현, 최지우

EWHA EURON

목차

- Traditional Methods for Machine Learning in Graphs
- Node-Level Tasks and Features
- Link Prediction Task and Features



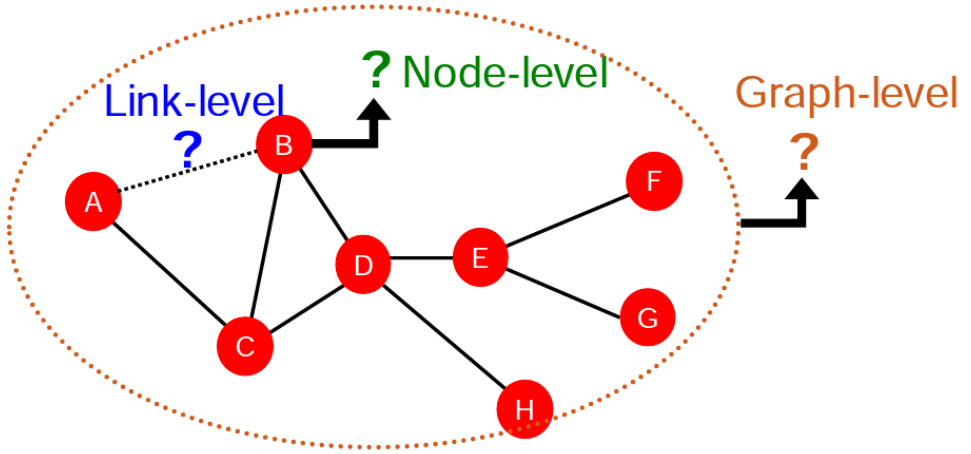
EWHA EURON

Traditional Methods for Machine Learning in Graphs

EWHA EURON

Traditional ML Pipeline

- Prediction의 범위
  - ☞ Node / Link / Graph Level



- 각 범위에 따른 feature를 적절하게 디자인

- 우리가 고려할 feature 2종류:

structural feature

nodes' attribute

→ 이번 강의에선 structural feature

- 모든 노드가 attribute를 각각 갖고 있다.

✚ Additional feature 찾아내서 -> more accurate prediction

Traditional ML Pipeline

STEP 1.

- feature vector 로 node/link/graph 표현

+

- ML 모델 훈련

STEP 2.

- 훈련된 모델 적용 (새로운 node/link/graph 에 적용 + 예측

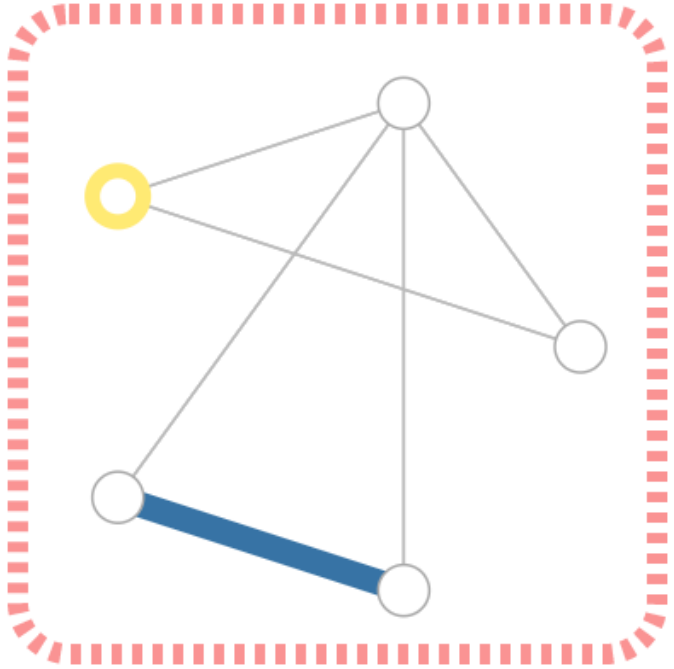
주된 목표: Feature Design

- graph에 Effective features 사용 → 모델 성능 향상

- traditional features for node-level/link-level/graph-level prediction
- undirected graphs 에 초점

Machine Learning in Graphs

- Given: $G = (V, E)$
- Learn a function: $f : V \rightarrow \mathbb{R}$ - objective function: 예측하고자 하는 label이 무엇인가?



Vertex (or node) embedding

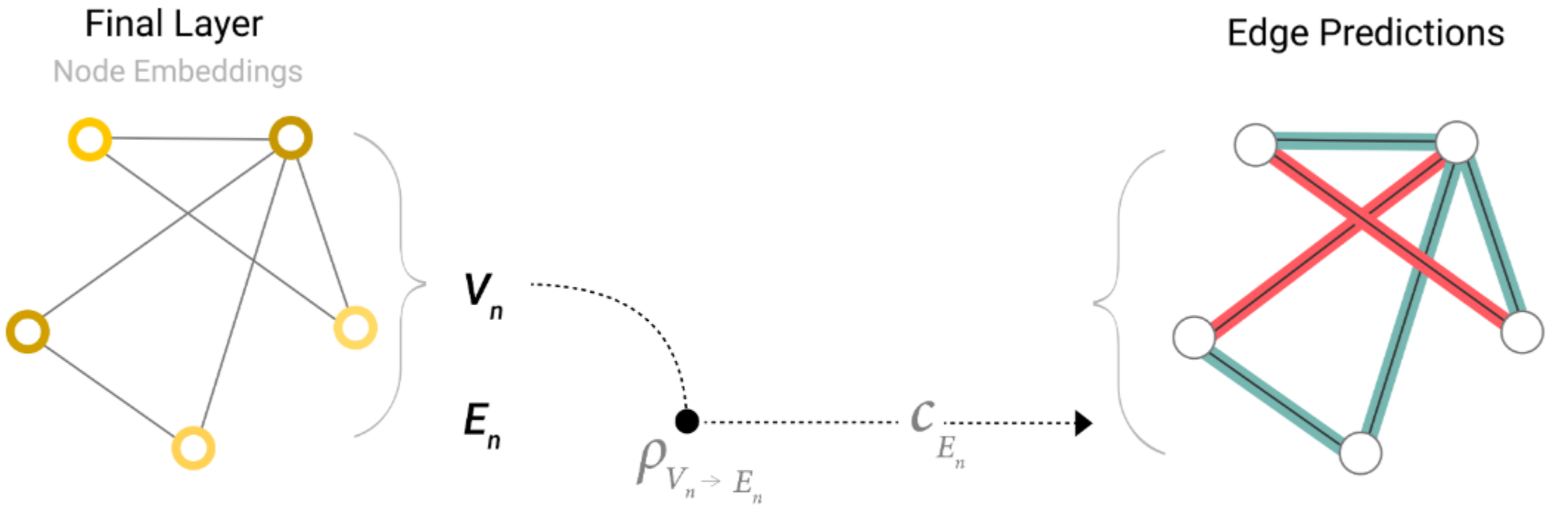
Edge (or link) attributes and embedding

Global (or master node) embedding

Information in the form of scalars or embeddings can be stored at each graph node (left) or edge (right).

https://distill.pub/2021/gnn-intro/

Machine Learning in Graphs



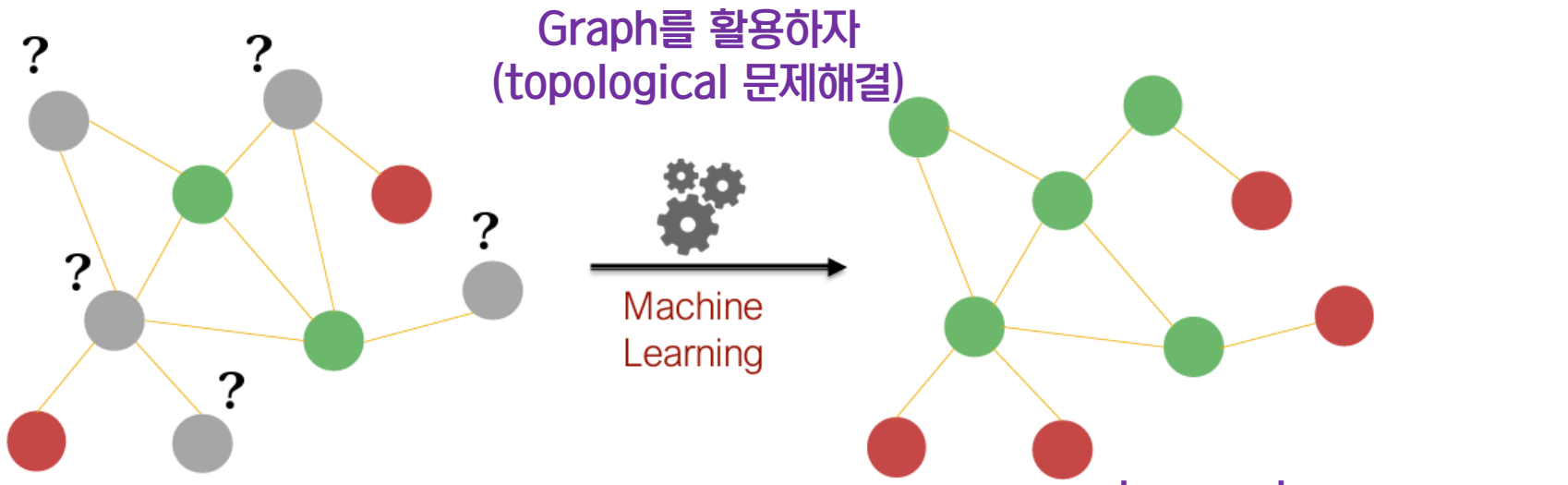
- node-level features를 갖고서,
  binary edge-level 정보를 예측하고자 하는 경우

https://distill.pub/2021/gnn-intro/

Node-Level Tasks and Features

EWHA EURON

Node-Level Tasks



Uncolored 노드 예측해보자

Node classification

패턴 도출해낼
feature 필요함

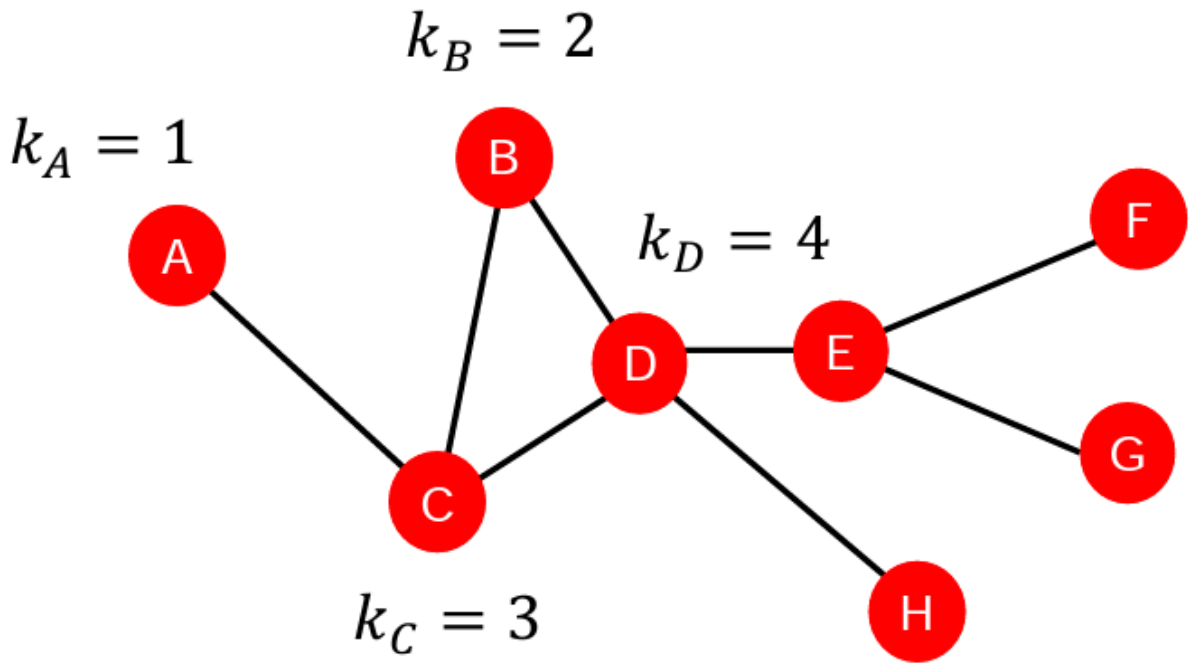
✔ 노드의 Feature 구하는 방법

- Node degree
- Node centrality
- Clustering coefficient
- Graphlets

Node-Level Features: Node Degree

- Node Degree: Node에 연결된 edge 수의 합 $$d_u = \sum_{v \in V} \mathbf{A}[u, v].$$

- 한계점: 이웃한 모든 nodes를 동일하게 취급함.



▶ C와 E처럼 degree값이 같은 node 를 똑같이 취급

Node-Level Features: Node Centrality

- **Node Centrality**: network 안에서 node 가 얼마나 중요한가?

  (network의 중심과 node 가 얼마나 가까운가)

- **Node Centrality 계산 방법**

  - Eigenvector centrality
  - Betweenness centrality
  - Closeness centrality

  etc...

Eigenvector Centrality

- 이웃한 nodes 중요도를 표현하는 척도
- import 이웃에 둘러 쌓여 있다면 ☞ 중요도 ⬆

$$c_v = \frac{1}{\lambda} \sum_{u \in N(v)} c_u \quad \Longleftrightarrow \quad \lambda \boldsymbol{c} = \boldsymbol{A}\boldsymbol{c}$$

λ is normalization const
(largest eigenvalue of A)

- $\boldsymbol{A}$: Adjacency matrix
  $\boldsymbol{A}_{uv} = 1$ if $u \in N(v)$
- $\boldsymbol{c}$: Centrality vector
- λ: Eigenvalue

Adjacency matrix 의 eigenvector ☞ centrality vector

Betweenness centrality

- 임의의 2개의 노드 간의 shortest path 위에 노드가 얼마나 자주 나타나는가?

  (둘 사이에 노드가 자주 있을수록 ☞ 중요도 UP!)

$$c_v = \sum_{s \neq v \neq t} \frac{\#(\text{shortest paths betwen } s \text{ and } t \text{ that contain } v)}{\#(\text{shortest paths between } s \text{ and } t)}$$

Closeness centrality

- ‘ 특정 노드 & 다른 모든 노드 ’ 간의 shortest path 길이

  (shortest path 길이가 짧을수록 ☞ 중요도 UP!)

$$c_v = \frac{1}{\sum_{u \neq v} \text{shortest path length between } u \text{ and } v}$$

Clustering Coefficient

- ‘ 특정 노드 & 이웃하는 노드들 ’ 얼마나 잘 연결되어 있는가

$$e_v = \frac{\#(\text{edges among neighboring nodes})}{\binom{k_v}{2}} \in [0,1]$$

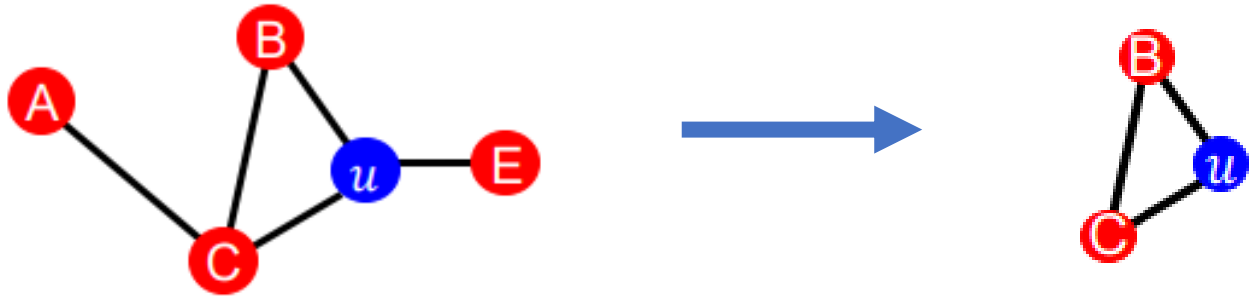
$$\#(\text{node pairs among } k_v \text{ neighboring nodes})$$

In our examples below the denominator is 6 (4 choose 2).

- ‘노드 v’ 의 이웃 노드들끼리의 노드쌍 경우의 수
- clustering coefficient =1 ☞ 모든 이웃 노드들이 서로 연결됨
- clustering coefficient =0 ☞ 모든 이웃 노드들 서로 연결 X

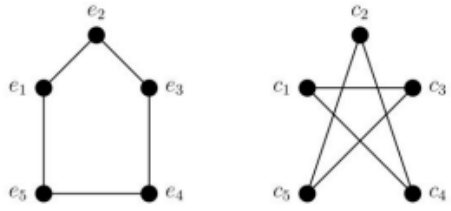
Graphlets

- 서로 연결되어 있는 induced non-isomorphic subgraph
- induced subgraph: 여러 노드들을 선택한 경우, 노드들 사이에 연결된 모든 edge 를 포함하는 subgraph



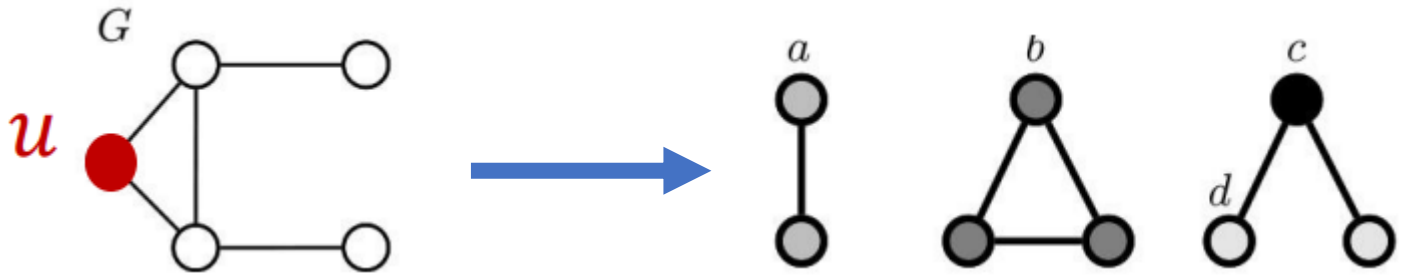
- graph isomorphism: 같은 수의 노드를 가지는 2개의 graph

(노드가 연결된 엣지 관계가 서로 같은 graph)

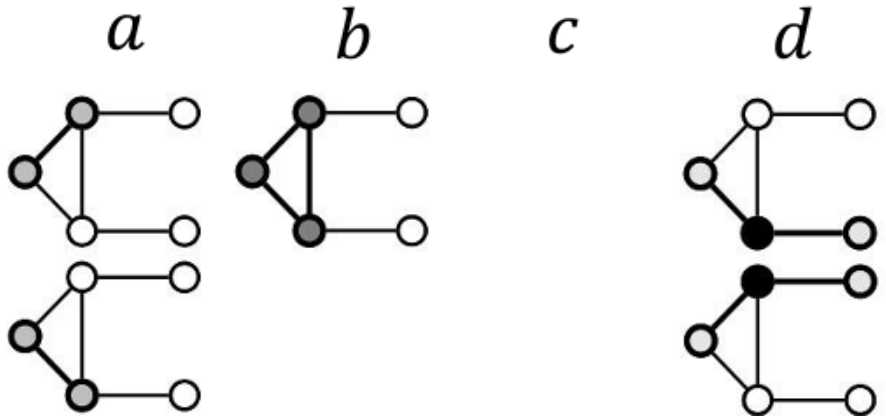


Graphlet Degree Vector (GDV)

- 한 노드에 대해 가능한 graphlet의 수를 의미하는 vector



Graphlet instances of node u:



GDV of node u:

a, b, c, d

[2,1,0,2]

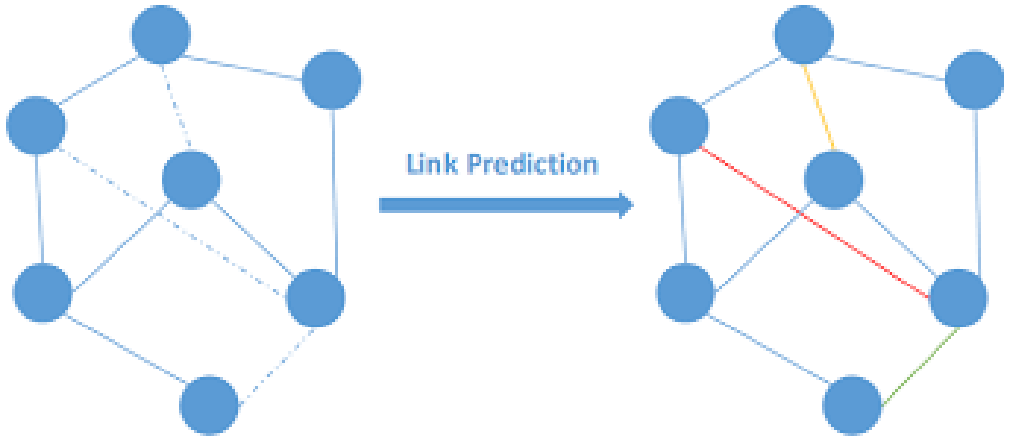
Link Prediction Task and Features

EWHA EURON

Link Prediction

- 핵심 : Node 쌍의 feature 디자인

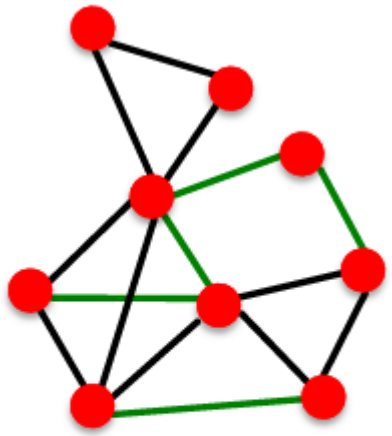
① 임의의 link 들을 제거한 다음, predict



② 시간의 흐름에 따라 발전하는 네트워크 (예: SNS, Collaboration network 등)

- 미래 시점의 그래프 예측

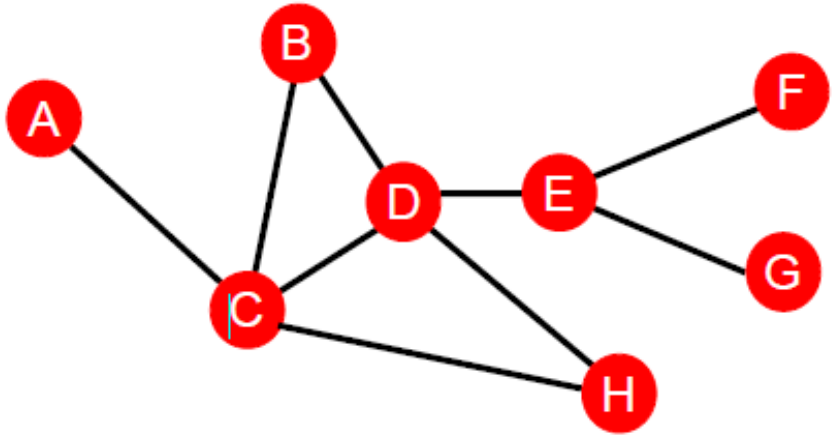
☞ 나중에 실제 발생한 그래프와 비교



$G[t_0, t_0']$

$G[t_1, t_1']$

The Shortest-path distance



$$S_{BH} = S_{BE} = S_{AB} = 2$$

$$S_{BG} = S_{BF} = 3$$

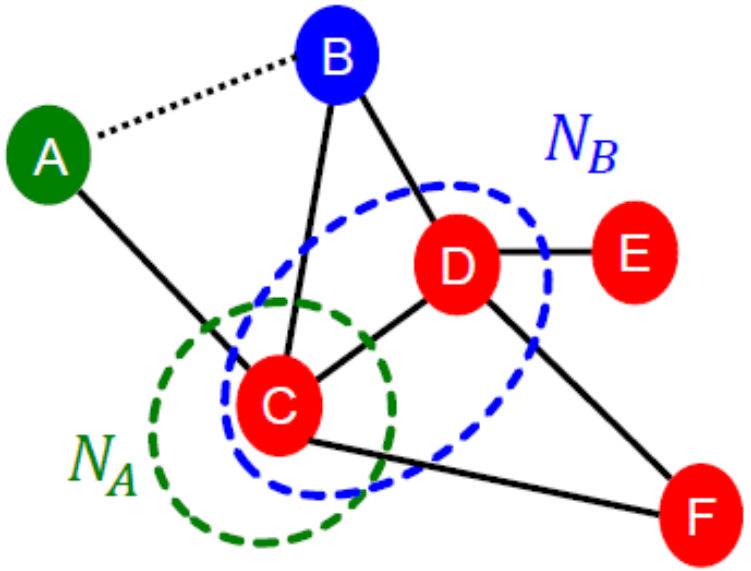
B,H 의 노드 쌍은 Shortest-path distance 2를 가지므로 neighboring node C, D를 갖는다고 볼 수 있다. 이때 neighborhood overlap의 차수는 측정하지 못하게 되는데 중첩된 neighborhood nodes를 찾기 위해 다양한 방법이 제기된다.

#Local Neighborhood Overlap

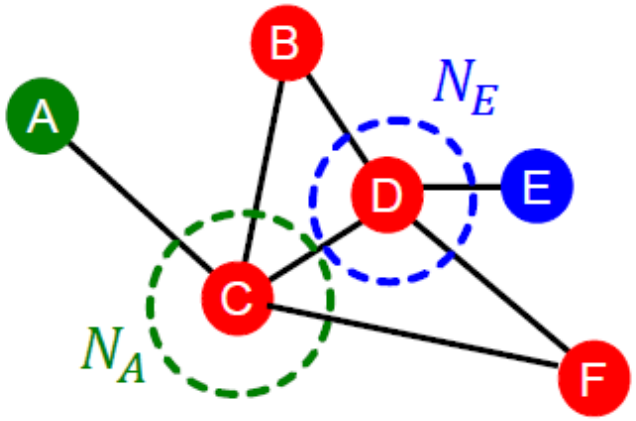
- **Common neighbors:** $|N(v_1) \cap N(v_2)|$
  - Example: $|N(A) \cap N(B)| = |\{C\}| = 1$
- **Jaccard's coefficient:** $\frac{|N(v_1) \cap N(v_2)|}{|N(v_1) \cup N(v_2)|}$
  - Example: $\frac{|N(A) \cap N(B)|}{|N(A) \cup N(B)|} = \frac{|\{C\}|}{|\{C,D\}|} = \frac{1}{2}$
- **Adamic-Adar index:**

  $\sum_{u \in N(v_1) \cap N(v_2)} \frac{1}{\log(k_u)}$

  - Example: $\frac{1}{\log(k_C)} = \frac{1}{\log 4}$



#Local Neighborhood Overlap



$$N_A \cap N_E = \phi$$
$$|N_A \cap N_E| = 0$$

두 nodes 사이에 공통 neighbor node가 없으면 0을 반환하는 한계를 갖는다. 두 노드가 추후에 잠재적인 연결이 있을 수도 있기 때문에

전체적인 graph을 고려하는 Global neighborhood overlap metrics는 이 한계를 해결할 수 있다.

#Global Neighborhood Overlap

Katz index : 모든 노드 쌍 사이의 모든 walks 수를 측정한 값
Walks는 vertexs, edges가 교대로 나타는 시퀀스

How to compute #walks between two nodes?

Use **adjacency matrix powers**!

- $\boldsymbol{A}_{uv}$ specifies #walks of length 1 (direct neighborhood) between u and v.
- $\boldsymbol{A}_{uv}^{2}$ specifies #walks of **length 2** (neighbor of neighbor) between u and v.
- And, $\boldsymbol{A}_{uv}^{\boldsymbol{l}}$ specifies #walks of **length $\boldsymbol{l}$**.

Katz index between v_1 and v_2 is calculated as

Sum over *all walk lengths*

$$S_{v_1 v_2} = \sum_{l=1}^{\infty} \beta^l \boldsymbol{A}^l_{v_1 v_2}$$

#walks of length l between v_1 and v_2

$0 < \beta < 1$: discount factor

Katz index matrix is computed in closed-form:

$$\boldsymbol{S} = \sum_{i=1}^{\infty} \beta^i \boldsymbol{A}^i = \underbrace{(\boldsymbol{I} - \beta \boldsymbol{A})^{-1}}_{= \sum_{i=0}^{\infty} \beta^i \boldsymbol{A}^i \text{ by geometric series of matrices}} - \boldsymbol{I},$$

#Global Neighborhood Overlap

Distance-based features:

-두 노드의 the shortest path length 를 이용. 중첩되는 neighborhood nodes를 capture 할 수 없는 한계가 있다.

Local neighborhood overlap:

-두 노드의 공유되는 neighboring nodes 개수를 계산할 수 있으며 이는 Distance-based features를 보완한 방법이다

-하지만 Local을 따지기 때문에 공유되지 않은 nodes의 값을 0으로 반환한다. 전체적인 그래프를 볼 때 잠재적인 connection이 있을 수 있기 때문에 한계가 발생한다.

Global neighborhood overlap:

-global graph structure 사용

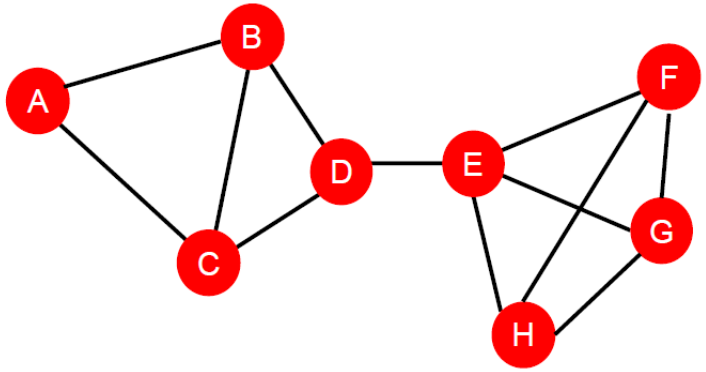
-katz index가 노드 쌍의 모든 walks를 계산

Graph-Level Features and Graph Kernels

EWHA EURON

#Graph-Level Features

For example:



전체의 graph의 구조를 어떻게 묘사할 수 있을까?
→ kernel Methods를 사용!

Idea: Design kernels instead of feature vectors.

A quick introduction to Kernels:

- Kernel $K(G, G') \in \mathbb{R}$ measures similarity b/w data
- Kernel matrix $\boldsymbol{K} = \left(K(G, G')\right)_{G,G'}$ must always be positive semidefinite (i.e., has positive eigenvalues)
- There exists a feature representation $\phi(\cdot)$ such that $K(G, G') = \phi(\mathrm{G})^{\mathrm{T}}\phi(G')$
- Once the kernel is defined, off-the-shelf ML model, such as kernel SVM, can be used to make predictions.

Feature vectors을 대신 쓰는 개념
Vectors의 내적값을 사용
Kenel matrix K는 대칭을 갖는 positive semidefinite가 되어야 한다.

M positive semi-definite $\iff$ $x^{\mathrm{T}}Mx \geq 0$ for all $x \in \mathbb{R}^n \setminus \mathbf{0}$

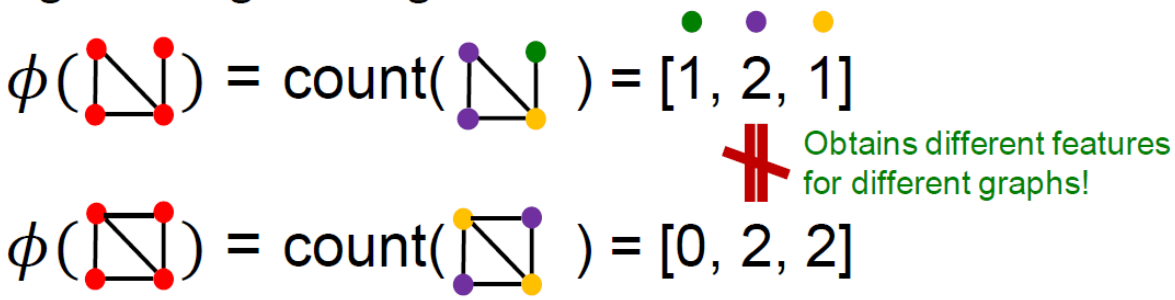
#Global kernel

Graph Kernels: Measure similarity between two graphs:

- Graphlet Kernel [1]
- Weisfeiler-Lehman Kernel [2]

What if we use Bag of **node degrees**?

Deg1: ● Deg2: ● Deg3: ●



Kernel은 Bag-of-Words(BOW)의 아이디어를 가져와서 graph feature vector를 구축한다.

BOW는 documents를 the word로 표현하여 계산하는데 graph에서는 words를 nodes로 생각한다.

이를 이용하여 다른 graph에도 같은 feature vector를 취할 수 있다.

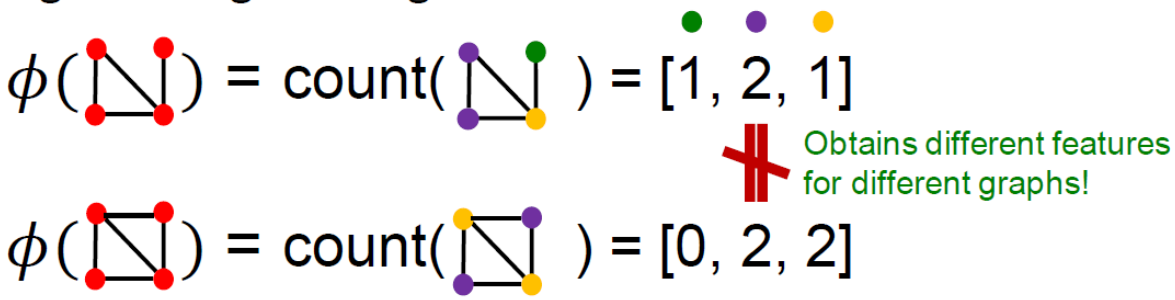
#Global kernel

Graph Kernels: Measure similarity between two graphs:

- Graphlet Kernel [1]
- Weisfeiler-Lehman Kernel [2]

What if we use Bag of **node degrees**?

Deg1: ● Deg2: ● Deg3: ●



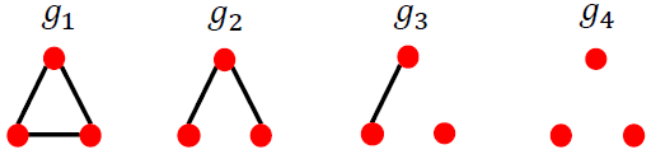
Kernel은 Bag-of-Words(BOW)의 아이디어를 가져와서 graph feature vector를 구축한다.

BOW는 documents를 the word로 표현하여 계산하는데 graph에서는 words를 nodes로 생각한다.

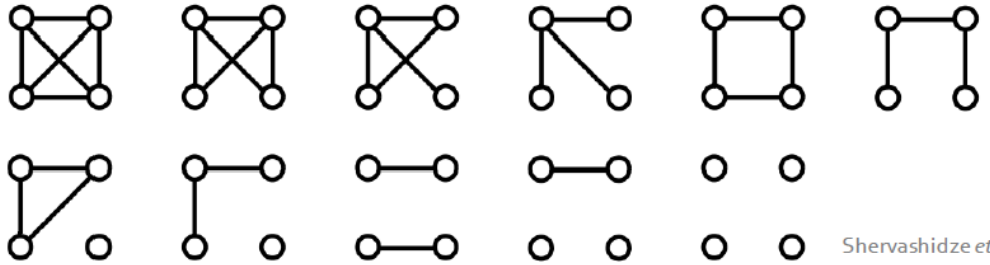
이를 이용하여 다른 graph에도 같은 feature vector를 취할 수 있다.

#Graphlet

For $k = 3$, there are 4 graphlets.



For $k = 4$, there are 11 graphlets.



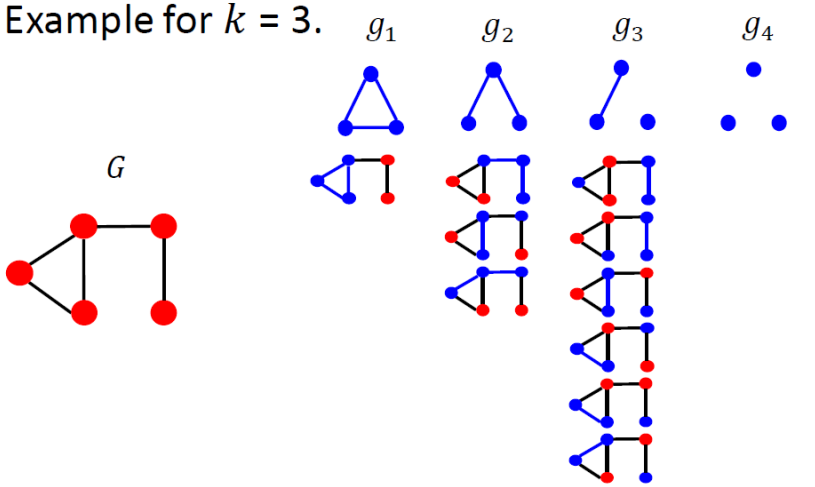
Shervashidze *et al.*

Graphlet은 주어진 node를 특징화할 수 있고 주어진 node 바탕으로 network 구조를 설명할 수 있다.

Node-level features와 다른 점은 graphlets는 뿌리가 없고 isolated nodes를 허용한다.

#Graphlet kernel

Example for k = 3.



$$f_G = (1, \quad 3, \quad 6, \quad 0)^{\mathrm{T}}$$

Given two graphs, G and G', graphlet kernel is computed as

$$K(G, G') = \boldsymbol{f}_G{}^{\mathrm{T}} \boldsymbol{f}_{G'}$$

Problem: if G and G' have different sizes, that will greatly skew the value.
Solution: normalize each feature vector

$$\boldsymbol{h}_G = \frac{\boldsymbol{f}_G}{\mathrm{Sum}(\boldsymbol{f}_G)} \qquad K(G, G') = \boldsymbol{h}_G{}^{\mathrm{T}} \boldsymbol{h}_{G'}$$

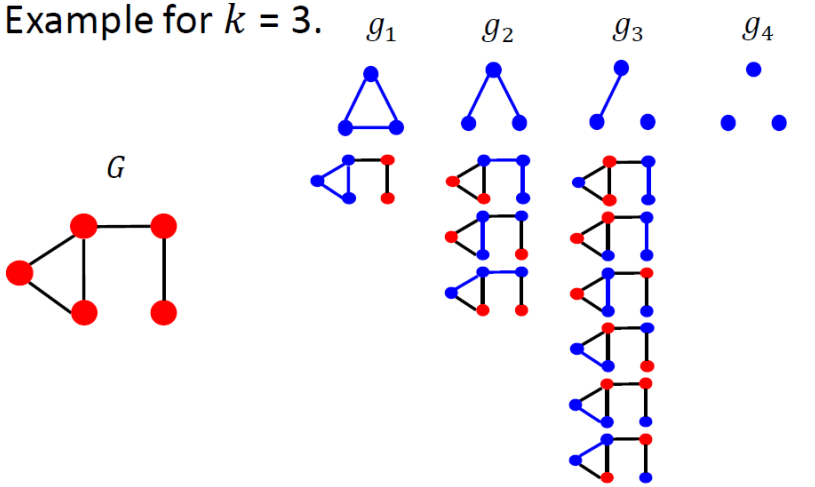
Graphlet count vector fG를 구하고 다른 두 그래프에 대해 내적하면 graphlet kernel을 계산할 수 있다.

Graphlets를 계산하는 것은 계산량이 많다는 한계를 갖는다. 또한 최악의 경우를 피할 수 없는 NP-Hard test이다.

→ 효과적인 graph kernel을 만들려면?
Weisfeiler-Lehman Kernel

#Graphlet kernel

Example for k = 3.



$$f_G = (1, \quad 3, \quad 6, \quad 0)^{\mathrm{T}}$$

Given two graphs, G and G', graphlet kernel is computed as

$$K(G, G') = \boldsymbol{f}_G{}^{\mathrm{T}} \boldsymbol{f}_{G'}$$

Problem: if G and G' have different sizes, that will greatly skew the value.
Solution: normalize each feature vector

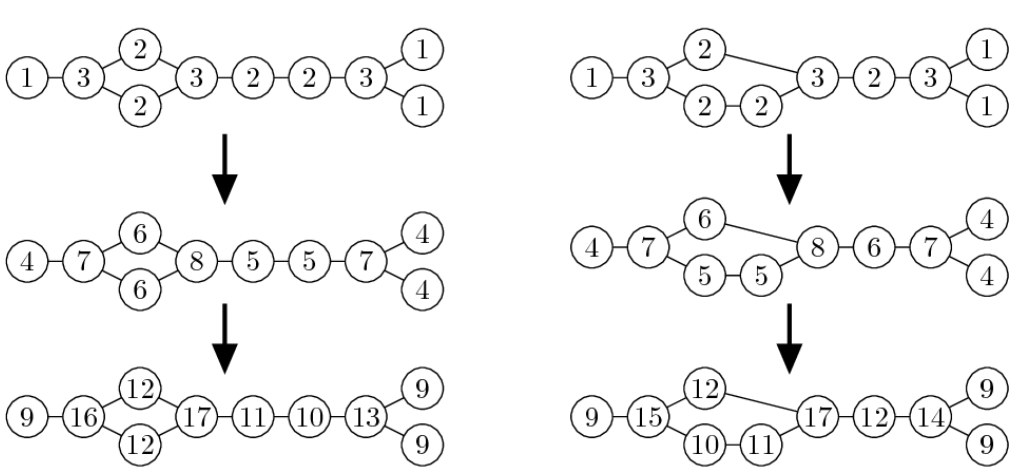
$$\boldsymbol{h}_G = \frac{\boldsymbol{f}_G}{\mathrm{Sum}(\boldsymbol{f}_G)} \qquad K(G, G') = \boldsymbol{h}_G{}^{\mathrm{T}} \boldsymbol{h}_{G'}$$

Graphlet count vector fG를 구하고 다른 두 그래프에 대해 내적하면 graphlet kernel을 계산할 수 있다.

Graphlets를 계산하는 것은 계산량이 많다는 한계를 갖는다. 또한 최악의 경우를 피할 수 없는 NP-Hard test이다.

→ 효과적인 graph kernel을 만들려면?
Weisfeiler-Lehman Kernel

#Weisfeiler-Lehman Kernel



histogram	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
G	3	4	3	3	2	2	2	1	3	1	1	2	1	0	0	1	1
G'	3	4	3	3	2	2	2	1	3	1	1	2	0	1	1	0	1

Neighborhood 구조를 이용해서 node vocabulary 더 풍부해지도록 반복한다. 이때 Color refinement 알고리즘을 사용한다.

WL kernel은 color count vectors의 내적 값을 통해 계산된다. WL kernel은 효과적이며 시간복잡성이 선형적이다.