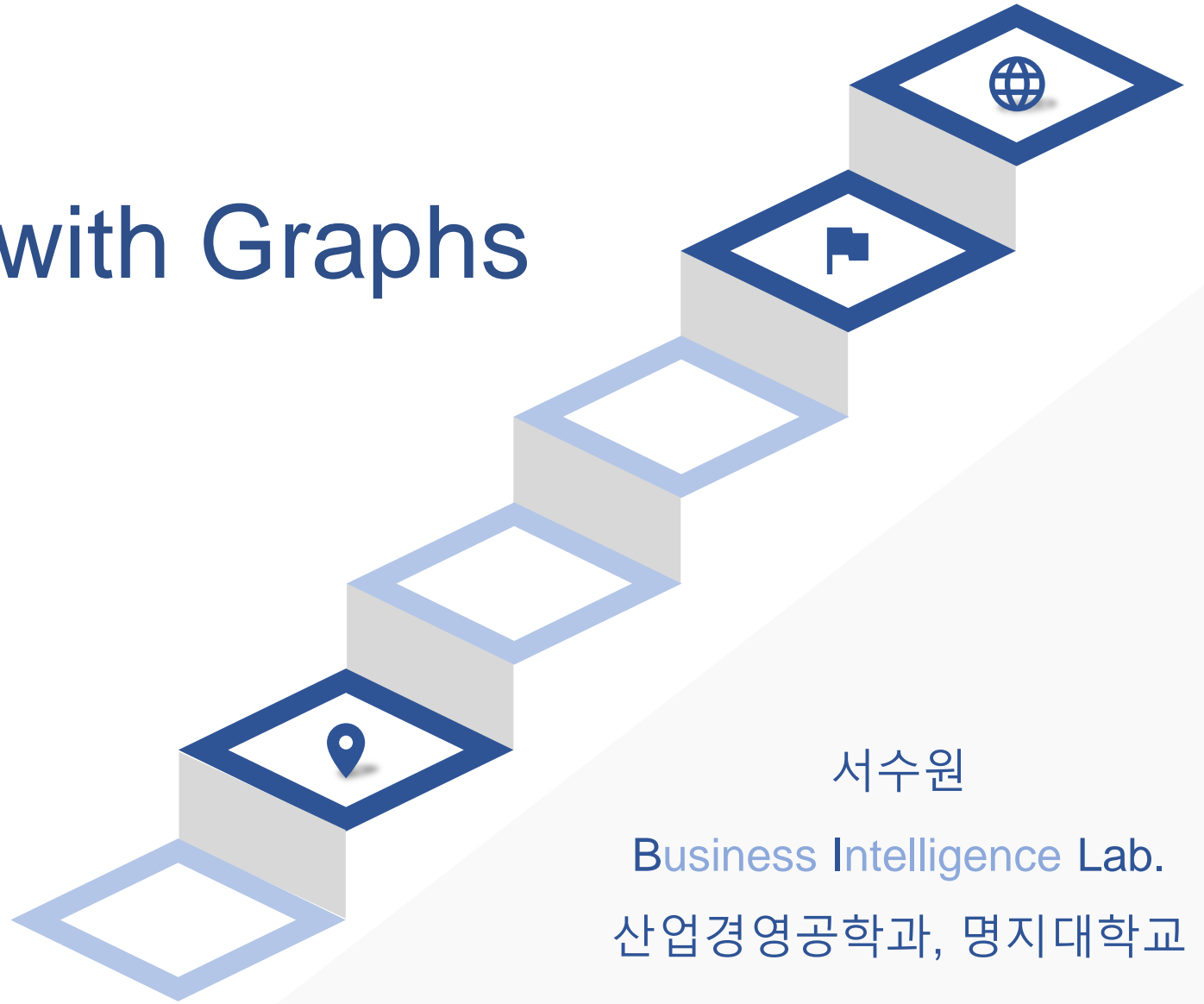




20230706

Machine Learning with Graphs



서수원

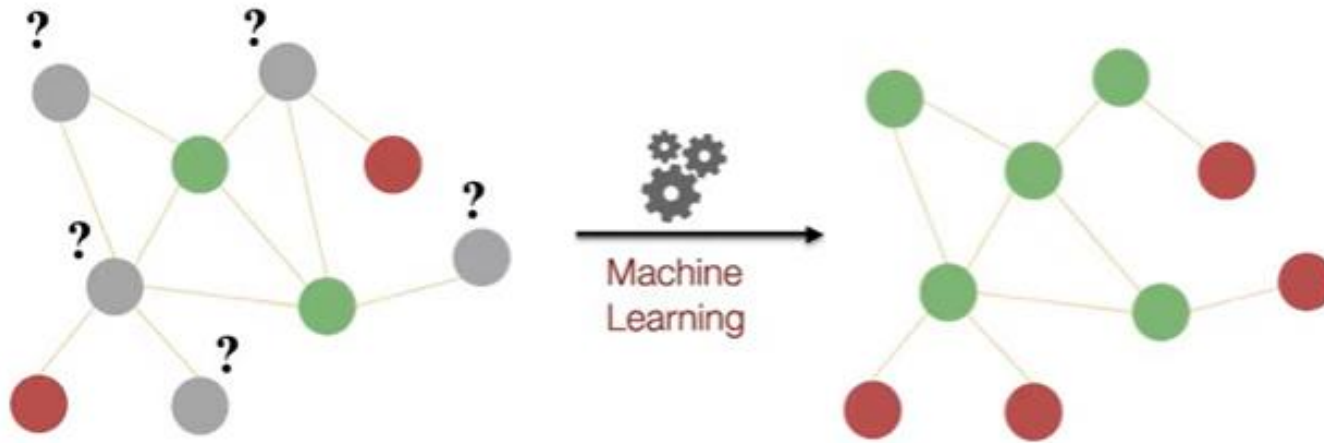
Business Intelligence Lab.
산업경영공학과, 명지대학교

01

**Node-level
prediction**

Node-level Tasks

- Node classification by Intuition
 - 빨간색 노드와 초록색 노드가 주어졌을 때 회색 노드는 어디에 속하는지를 알고 싶다.
 - ✓ 빨간색 노드는 하나의 엣지만 있고 녹색 노드는 두개 이상의 엣지가 있다.
 - ❖ 이는 노드의 차수를 통해 회색 노드의 레이블을 예측 할 수 있음을 의미한다.



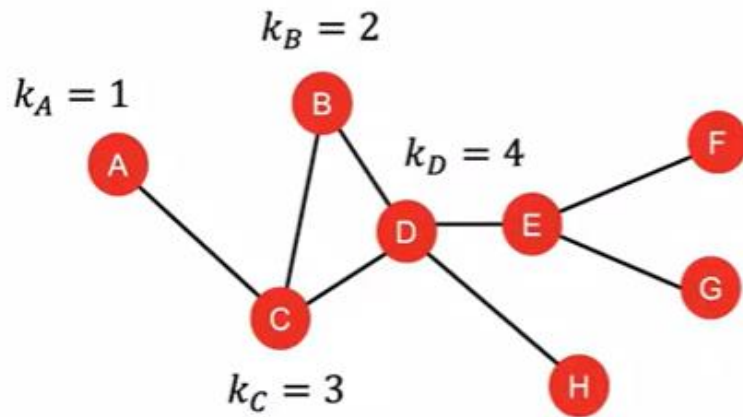
Node classification

Node-level Overview

- Goal
 - 예측을 위해서는 그래프에서 특징을 뽑아내어 유클리드 공간 또는 벡터에 투영해야 한다.
 - Given: $G = (V, E)$
 - Learn a function: $f : V \rightarrow \mathbb{R}$
- Node Feature
 - Node degree
 - Node centrality
 - Clustering coefficient
 - graphlets

Node Degree

- Node Degree
 - 노드의 차수는 노드가 가진 엣지의 수를 의미한다.
 - 단점
 - ✓ 모든 이웃 노드를 가중치 없이 동등히 취급한다.
 - ❖ 가중치를 줄 수 없다는 의미이다.
 - ✓ 같은 차수의 노드는 같은 레이블이라고 인식한다.
 - ❖ H,A,F,G를 구분 할 수 없다.



Node Centrality

- Node Centrality
 - 노드 중심성은 노드의 중요도를 확인 하는 것이 목적이다.
- Different ways to model importance
 - Eigenvector centrality
 - Betweenness centrality
 - Closeness centrality

Node Centrality - Eigenvector

- Eigenvector centrality(고유벡터 중심성)
 - 노드 v 가 중요이웃인 노드 u 들과 엣지가 있다면 중요하다고 판단한다.

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

λ is some positive constant

- \mathbf{A} : Adjacency matrix
 $A_{uv} = 1$ if $u \in N(v)$
- \mathbf{c} : Centrality vector

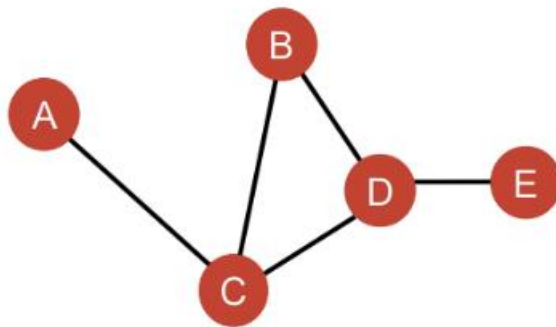
- ✓ V 의 중요도는 이웃의(u 들의) 중요도의 합이다.
- ✓ A 는 인접 행렬이다.(N by N)
- ✓ C 는 중심성 정도이다. (1 by N)
- ✓ 람다는 정규화를 위한 것인데, 음수가 아닌 상수이다.
 - ❖ 방향성이 없기 때문이다.
 - ❖ 오른쪽 수식을 행렬방정식을 통해 계산하면 람다는 N 개가 가능하고 그중 최대값의 람다를 이용한다.
 - » 그래야 C 값도 최대값으로 되기 때문이다.

Node Centrality - Betweenness

- Betweenness centrality
 - 노드의 중요성은 다른 노드들 간의 최단 경로에 얼마나 속해 있는 지에 따라 결정된다.
 - ✓ 노드를 임의로 두개 선택 했을 때 다른 노드 t 가 선택한 노드의 최단경로에 포함 되는 수가 수식의 분자이고, 임의 선택한 노드의 모든 최단 경로의 수가 분자이다. 그 모든 값을 더하면 C_t 가 나온다.

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

■ Example:



$$c_A = c_B = c_E = 0$$
$$c_C = 3$$

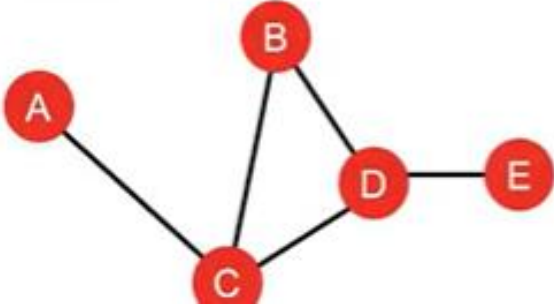
(A-C-B, A-C-D, A-C-D-E)

$$c_D = 3$$

(A-C-D-E, B-D-E, C-D-E)

Node Centrality - Closeness

- Closeness centrality
 - 노드가 얼마나 그래프의 중앙에 있는지를 수식화 한 것이다.
 - ✓ c_v 를 구할 때 임의의 다른 노드와의 최단 거리의 합을 분모로 보고 계산한다.

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


$c_A = 1/(2 + 1 + 2 + 3) = 1/8$
(A-C-B, A-C, A-C-D, A-C-D-E)

$c_D = 1/(2 + 1 + 1 + 1) = 1/5$
(D-C-A, D-B, D-C, D-E)

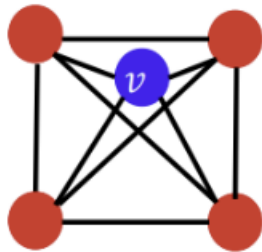
Clustering Coefficient

- Clustering Coefficient

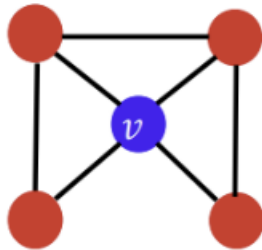
- 분자는 인접 노드 사이의 엣지의 수를 의미한다.
- 분모의 K_v 는 v 의 차수이다.
 - ✓ K_v combination 2를 하는 이유는 v 와 관계가 있는 이웃들이 연결되는 경우의 수를 의미한다.

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

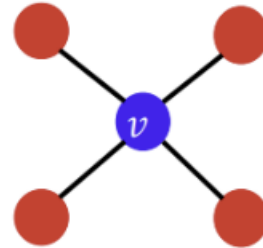
#(node pairs among k_v neighboring nodes)



$$e_v = 1$$



$$e_v = 0.5$$

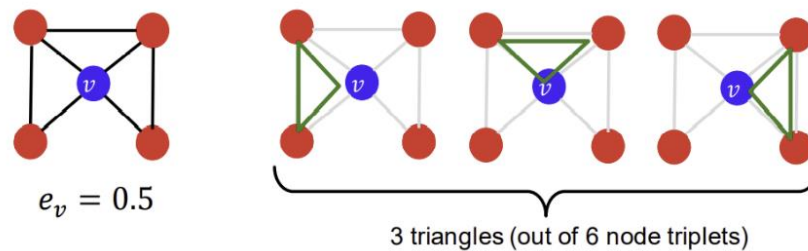


$$e_v = 0$$

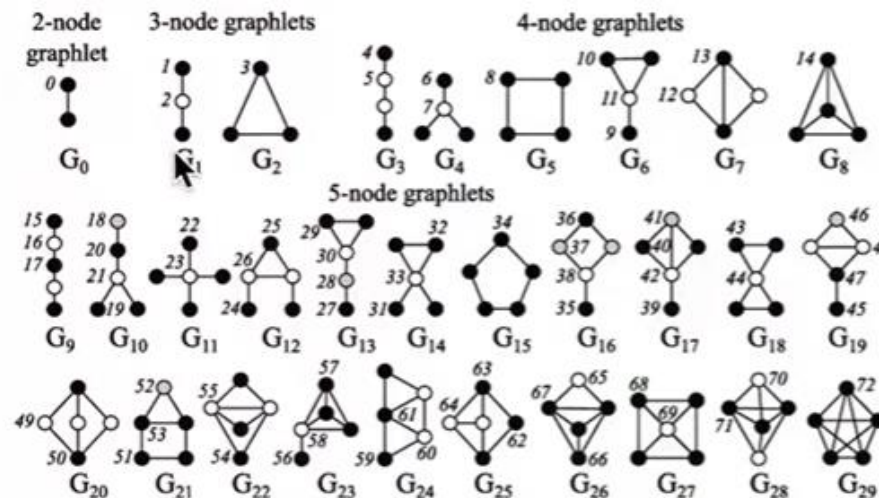
Graphlets

- Graphlets

- Clustering Coefficient를 Graphlets의 개념으로 일반화 할 수 있다.
- Clustering Coefficient는 기본적으로 인접 노드들 과의 삼각형을 세는 것이라고 볼 수 있다.



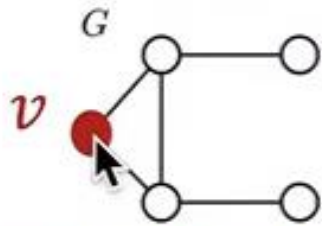
- Graphlets은 삼각형 직선 사각형 등 다양한 구조에 대해 다룬다.



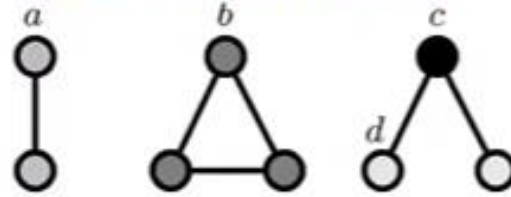
Graphlets

- Graphlets Degree Vector
 - Graphlets-based features for nodes
 - 노드가 가지는 graphlets 수를 센다.

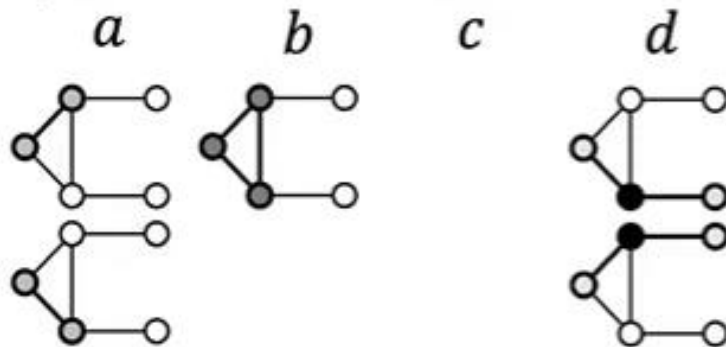
Example:



List of graphlets



Graphlet instances:



GDV of node v :

a, b, c, d

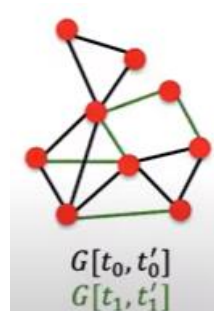
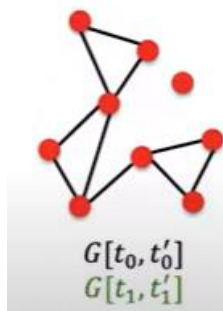
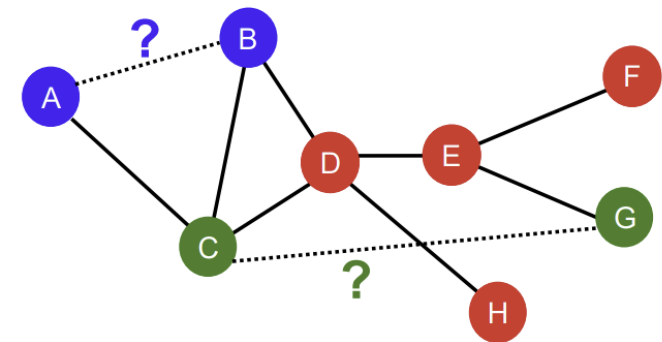
$[2, 1, 0, 2]$

02

Link-Level Prediction Task

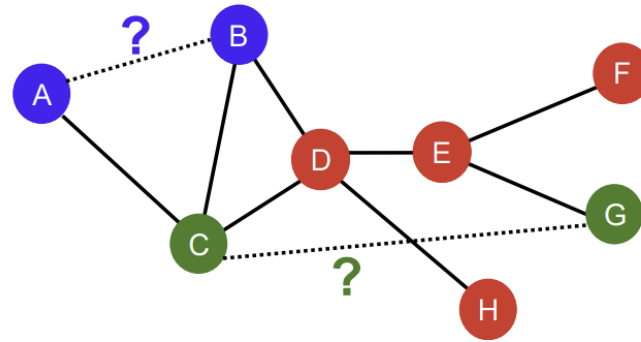
Link-level Tasks

- Link(Edge)-level Tasks
 - 기존 링크를 통해 새로운 링크를 예측 하는 문제를 의미한다.
- Two formulation of the link prediction Tasks
 - Links missing at random
 - ✓ 무작위로 링크를 삭제하고 링크를 예측한다.
 - Links over time
 - ✓ 시간에 따라 진화하는 네트워크의 경우 다음 시간의 링크를 예측한다.
 - ❖ Sns의 follow 수, 특허 링크



Link-level Tasks

- Features
 - Distance-based feature
 - Local neighborhood overlap
 - Global neighborhood overlap

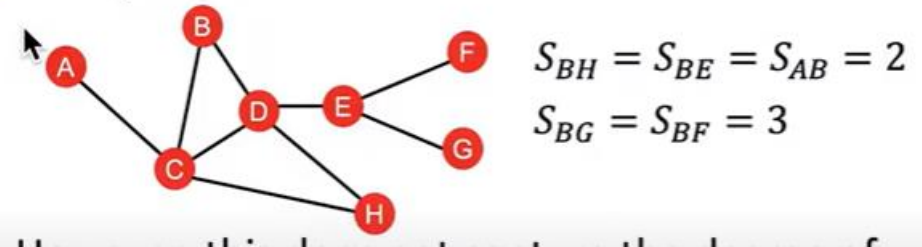


Distance-Based Features

- Distance-Based Features

- 두 노드 사이의 최단 거리를 구한다.
- **최단경로를 구분하지 않는다.**
 - ✓ S_{bh} 는 2개의 경로가 있고, S_{be} 는 하나의 경로만 있는데(연결강도의 차이가 있다.) 구분을 안한다는 의미이다.

■ Example:



Local Neighborhood Overlap

- Local Neighborhood Overlap

- 지역적으로 overlap을 고려할 수 있는 방법을 의미한다.

- ✓ 두 노드 사이의 연결강도를 구하기 위해 노드 사이에 직접 공유하는 노드를 잡는 방법이다.

- **Common neighbors:** $|N(v_1) \cap N(v_2)|$

- Example: $|N(A) \cap N(B)| = |\{C\}| = 1$

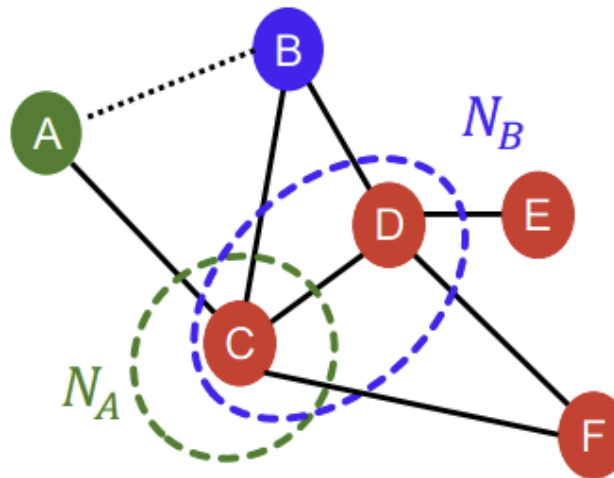
- **Jaccard's coefficient:** $\frac{|N(v_1) \cap N(v_2)|}{|N(v_1) \cup N(v_2)|}$

- Example: $\frac{|N(A) \cap N(B)|}{|N(A) \cup N(B)|} = \frac{|\{C\}|}{|\{C, D\}|} = \frac{1}{2}$

- **Adamic-Adar index:**

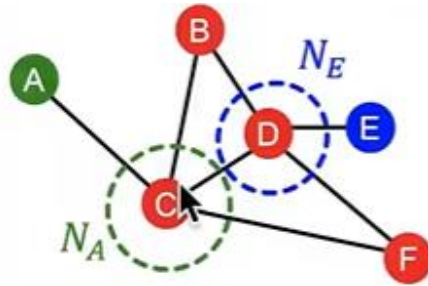
$$\sum_{u \in N(v_1) \cap N(v_2)} \frac{1}{\log(k_u)}$$

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



Global Neighborhood Overlap

- Global Neighborhood Overlap
 - Local Neighborhood Overlap의 경우 공통이웃이 존재하지 않으면 값이 0이 된다.



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

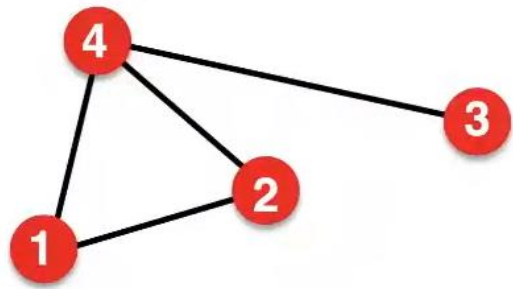
- 미래에 둘 사이의 connection이 생길 수 있는데 이를 완전히 무시한다는 의미이다.

Global Neighborhood Overlap

- Global Neighborhood Overlap
 - Katz index를 사용한다.
 - ✓ 주어진 노드 쌍 사이의 모든 길이의 가능한 수를 계산한다.
 - ❖ 인접행렬을 통해 계산이 가능하다.

Global Neighborhood Overlap

- Global Neighborhood Overlap : Computing #paths between tow nodes
 - 두 노드 사이의직접연결



$P_{12}^{(1)} = A_{12}$

$$A = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

Recall: $A_{uv} = 1$ if $u \in N(v)$
Let $P_{uv}^{(K)}$ = #paths of length K between u and v
We will show $P^{(K)} = A^k$

$P_{uv}^{(2)} = \sum_i A_{ui} * P_{iv}^{(1)} = \sum_i A_{ui} * A_{iv} = A_{uv}^2$

Node 1's neighbors #walks of length 1 between Node 1's neighbors and Node 2 $P_{12}^{(2)} = A_{12}^2$

Power of adjacency

$$A^2 = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \times \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 2 & 1 & 1 & 1 \\ 1 & 2 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 3 \end{pmatrix}$$

Global Neighborhood Overlap

- Global Neighborhood Overlap
 - Katz index를 사용한다.
 - ✓ 주어진 노드 쌍 사이의 모든 길이의 가능한 수를 계산한다.
 - ❖ 인접행렬을 통해 계산이 가능하다.

- A_{uv} specifies #paths of length 1 (direct neighborhood) between u and v .
- A_{uv}^2 specifies #paths of **length 2** (neighbor of neighbor) between u and v .
- And, A_{uv}^l specifies #paths of **length l** .

■ **Katz index** between v_1 and v_2 is calculated as

Sum over all walk lengths

$$S_{v_1 v_2} = \sum_{l=1}^{\infty} \boxed{\beta^l} \boxed{A_{v_1 v_2}^l} \begin{matrix} \text{\#walks of length } l \\ \text{between } v_1 \text{ and } v_2 \end{matrix}$$

$0 < \beta < 1$: discount factor

■ Katz index matrix is computed in closed-form:

$$S = \sum_{i=1}^{\infty} \beta^i A^i = \underbrace{(I - \beta A)^{-1}}_{= \sum_{i=0}^{\infty} \beta^i A^i} - I,$$

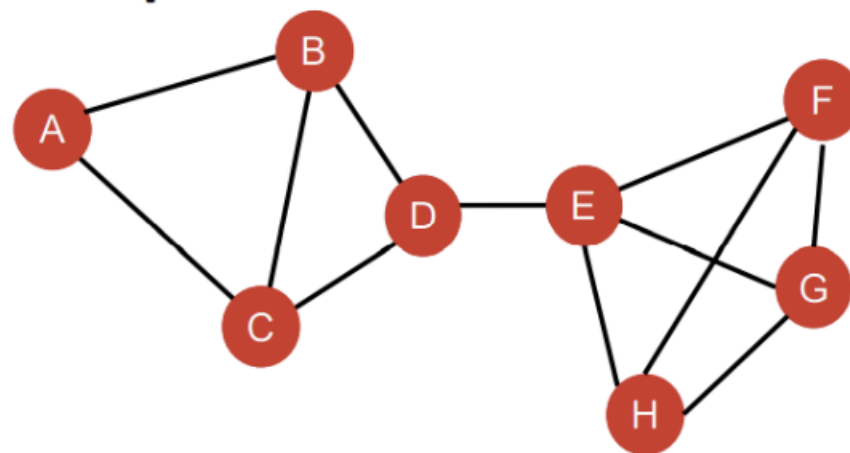
by geometric series of matrices

03

**Graph-Level
Prediction Task**

- Goals
 - 전체 그래프의 구조에 대한 특징을 찾는 것이 목적이다.

■ For example:



Graph-level Tasks

- Background-kernel methods
 - 특정 벡터가 아닌 커널을 설계하는 것을 의미한다.
- Introduction to kernels
 - Kernel $K(G, G') \in \mathbb{R}$ measures similarity b/w data
 - ✓ 두 그래프 사이의 유사도를 측정한다는 의미이다
 - Kernel matrix $\mathbf{K} = K(G, G')$ must always be positive semidefinite (i.e., has positive eigenvalues)
 - ✓ \mathbf{K} 은 항상 양수이다.
 - There exists a feature representation $\phi(\cdot)$ such that $K(G, G') = \phi(G)^T \phi(G')$ (-> dot product)
 - ✓ feature vector가 요청되는 임의의 함수 ϕ 가 존재한다.
 - Once the kernel is defined, off-the-shelf ML model, such as kernel SVM, can be used to make predictions.
 - ✓ kernel만 정의하면 ML이 가능하다.

- Graph Kernel : Key Idea

- BOW의 개념을 사용한다.
- 그래프를 구성하는 요소의 개수를 센다.
 - ✓ 단어 = node로 인식하면, 분명 다른 그래프 임에도, kernel method에 의해 같은 그래프로 판단된다.
 - ❖ 단순히 노드가 4개이기 때문에 같다고 인식한다.
 - ✓ 단어 = link로 인식하면 두 그래프의 다름을 인식한다.

$$\phi(\text{Graph 1}) = \phi(\text{Graph 2})$$

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

Deg1: ● Deg2: ● Deg3: ●

$$\phi(\text{Graph 1}) = \text{count}(\text{Graph 1}) = [1, 3, 0]$$

$$\phi(\text{Graph 2}) = \text{count}(\text{Graph 2}) = [0, 2, 2]$$

Obtains different features for different graphs!

Graph-level Tasks

Graphlet Kernel

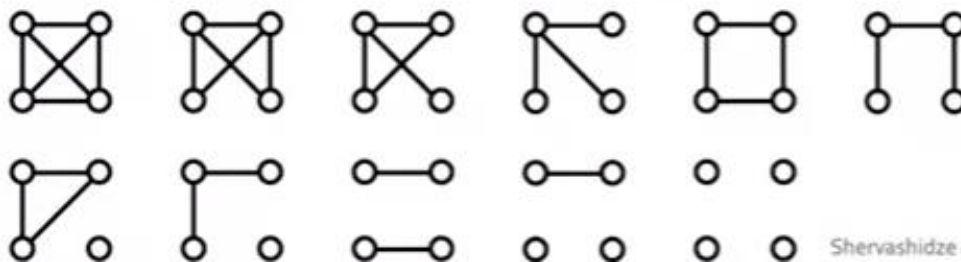
- 노드 레벨의 graphlet과는 다른점이 있다.
 - 무조건 노드들이 연결될 필요는 없다.
 - 고정된 노드가 필요 없다.

Let $\mathcal{G}_k = (g_1, g_2, \dots, g_{n_k})$ be a list of graphlets of size k .

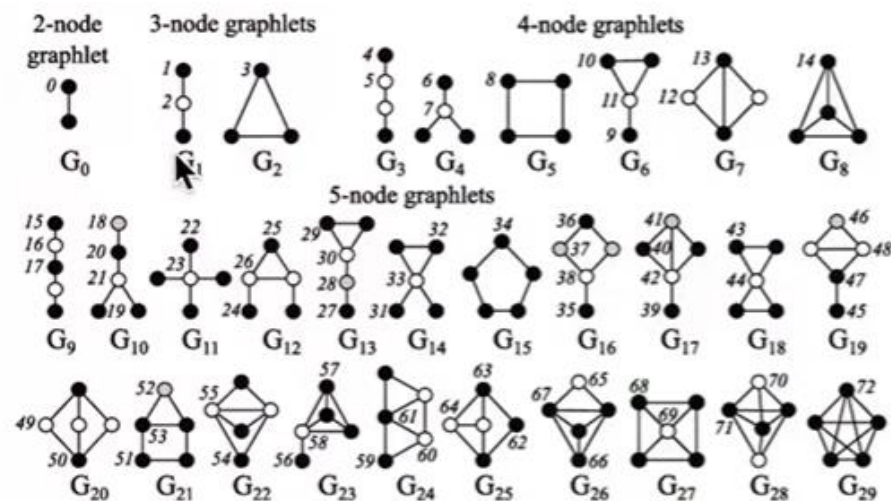
- For $k = 3$, there are 4 graphlets.



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



Shervashidze et al., AISTATS 2011

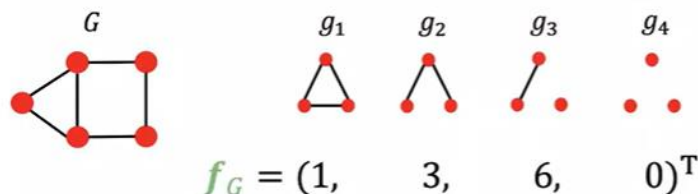


Graph-level Tasks

- Graphlet Kernel

- 그래프 G 와 graphlet list가 주어지면 graphlet count vector를 정의 할 수 있다.

■ Example for $k = 3$



- 두개의 그래프가 주어지면 내적을 통해 계산이 가능하다.

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

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

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

$$\mathbf{h}_G = \frac{\mathbf{f}_G}{\text{Sum}(\mathbf{f}_G)} \quad K(G, G') = \mathbf{h}_G^T \mathbf{h}_{G'}$$

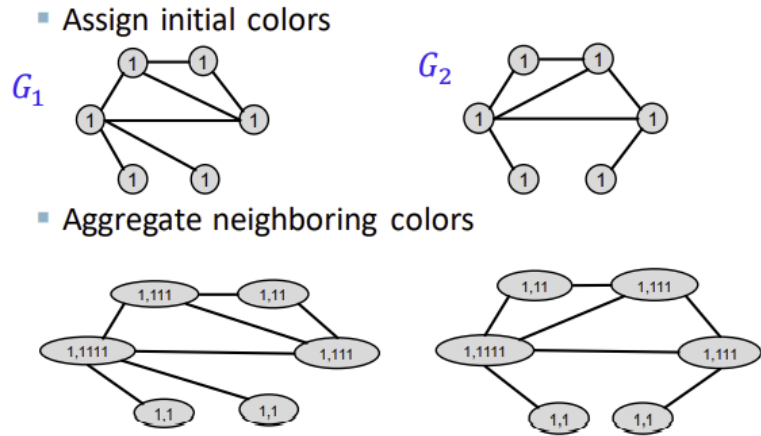
Graph-level Tasks

- Graphlet Kernel : Limitations
 - 계산이 너무 어렵다.
 - ✓ 비효율적이다.
 - ✓ 지수적으로 계산량이 늘어난다.
- Weisfeiler – Lehman Kernel
 - 비효율을 해결 했다.
 - 이웃 구조를 이용하여 반복적으로 그래프를 풍부하게 바꾼다.
 - **Given:** A graph G with a set of nodes V .
 - Assign an initial color $c^{(0)}(v)$ to each node v .
 - Iteratively refine node colors by
$$c^{(k+1)}(v) = \text{HASH} \left(\left\{ c^{(k)}(v), \{c^{(k)}(u)\}_{u \in N(v)} \right\} \right),$$
where **HASH** maps different inputs to different colors.
 - After K steps of color refinement, $c^{(K)}(v)$ summarizes the structure of K -hop neighborhood

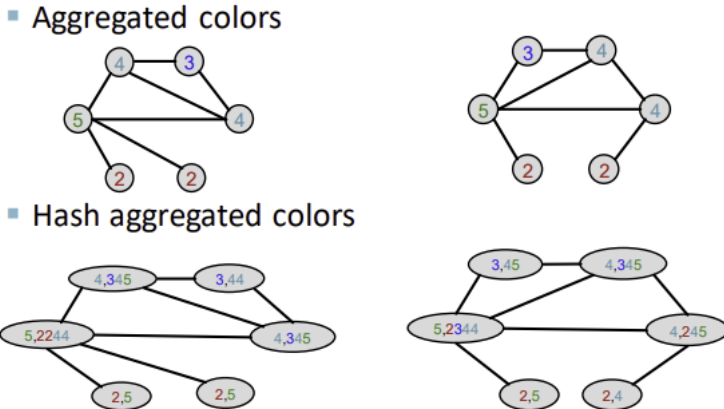
Graph-level Tasks

- Weisfeiler – Lehman Kernel

Example of color refinement given two graphs



Example of color refinement given two graphs



After color refinement, WL kernel counts number of nodes with a given color.

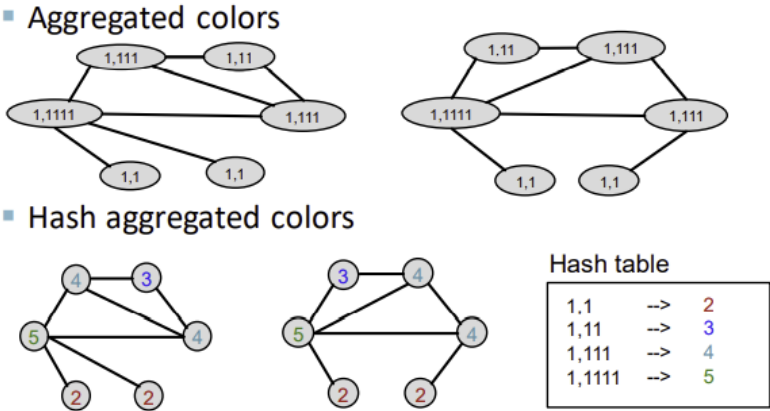
$\phi(\text{Graph 1}) = [6, 2, 1, 2, 1, 0, 2, 1, 0, 0, 0, 0, 2, 1]$

Colors: 1,2,3,4,5,6,7,8,9,10,11,12,13
Counts

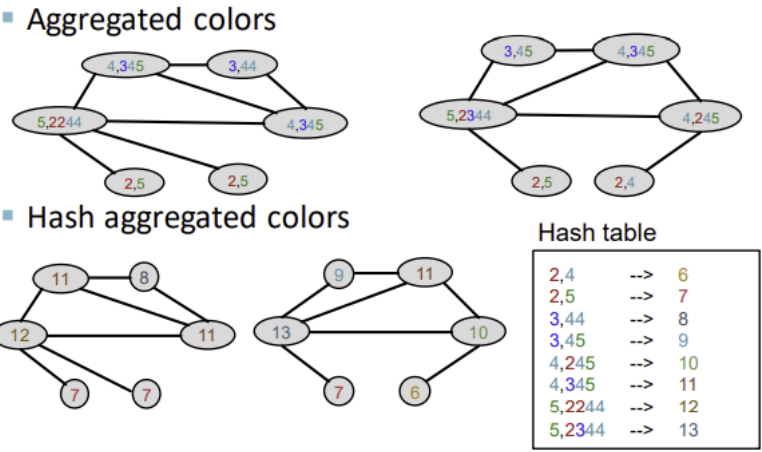
$\phi(\text{Graph 2}) = [6, 2, 1, 2, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1]$

Colors: 1,2,3,4,5,6,7,8,9,10,11,12,13
Counts

Example of color refinement given two graphs



Example of color refinement given two graphs



$K(\text{Graph 1}, \text{Graph 2}) = \phi(\text{Graph 1})^T \phi(\text{Graph 2}) = 49$



- Weisfeiler – Lehman Kernel
 - 시간 복잡도가 단순히 edge에 선형으로 늘어나기 때문에 계산에 효율이 생겼다.