# 2. Traditional Methods for Machine Learning in Graphs

https://www.youtube.com/watch?v=3IS7UhNMQ3U

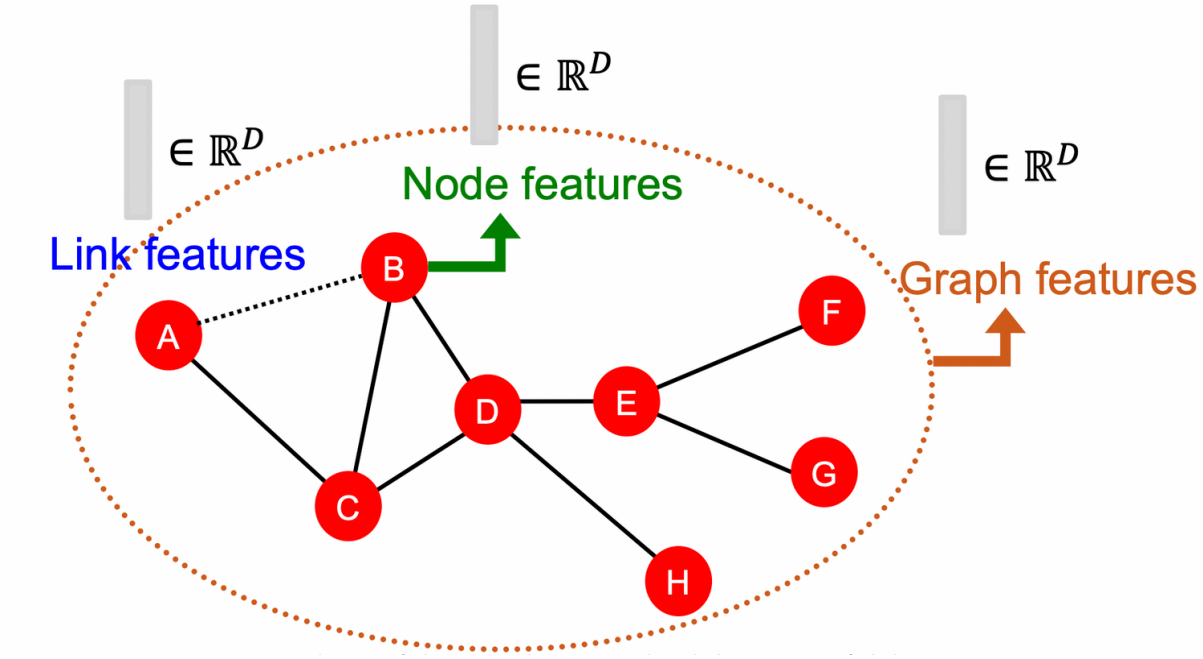
## Contents

1. Traditional Node-Level Tasks and Features
2. Traditional Edge-Level Tasks and Features
3. Traditional Graph-Level Features and Graph Kernels

---

## 1. Traditional Node-Level Tasks and Features

**Traditional ML Pipeline is about designing proper features!**

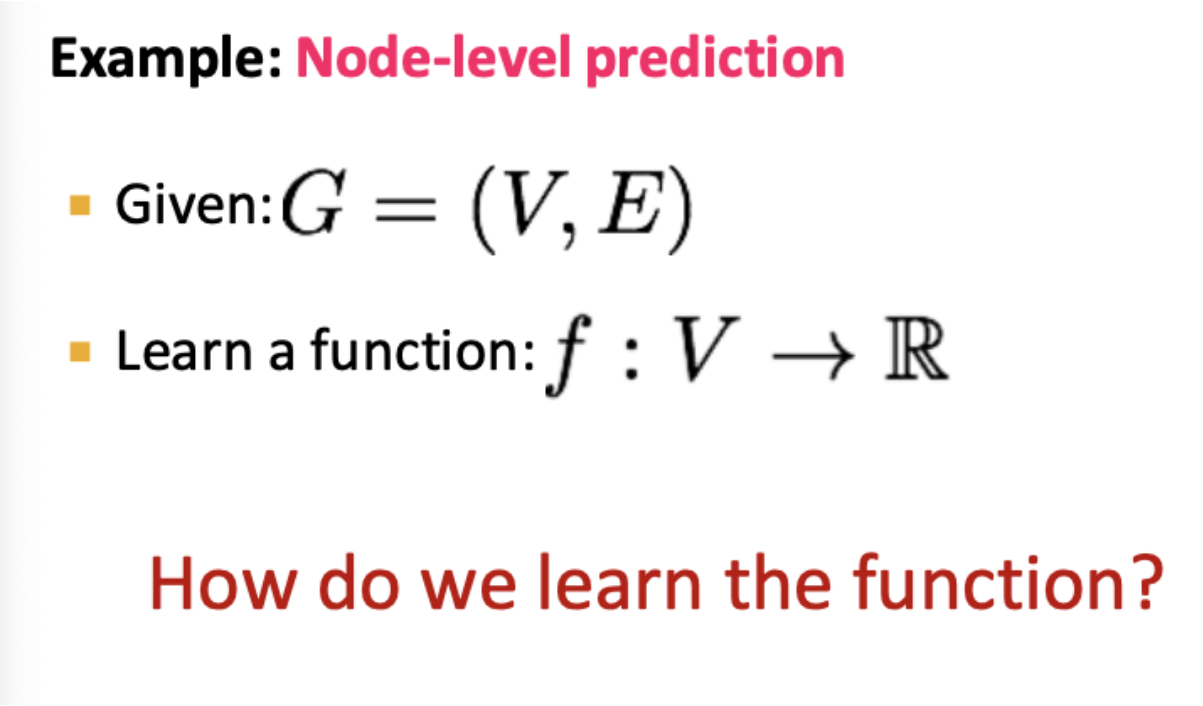



- Traditional ML : hadn-designed features을 이용
- Design features for nodes/links/graphs ⇒ 효과적인 feature representation 을 찾는 것이 performance ↑

1. structural feature(ex. positional...) → Graph-level prediction
2. attributes and properties of node → Node / Link level prediction

## Machine Learning in Graphs

- Goal : Make predictions for a set of objects
- Design choices:
  - Features : *d*-dimensional vectors
  - Objects : Nodes, edges, sets of nodes, entire graphs
  - Objective function : What task are we aiming to solve?(풀고자 하는 문제에 따라 다름)




---

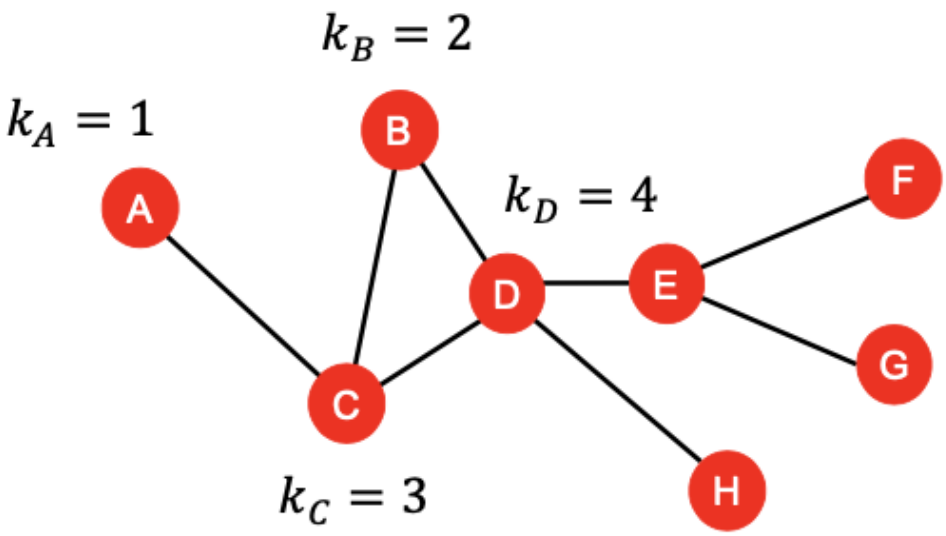
## Node-level Tasks and Features

Goal : **Characterize the structure and position of a node in the network using**

- Node degree
- Node centrality
- Clustering coefficient
- Graphlets

1. **Node degree**

- The degree $k_v$ of node v is t**he number of edges**(neighboring nodes) the node has.
- Treats all neighboring nodes equally.




- Node degree counts the neighboring nodes without capturing their importance.
- **Node centrality** $c_v$ takes the node importance in a graph into account

2. **Node Centrality**

- 노드 v의 이웃 노드의 centrality의 합이 크면 importance ↑
- Node Centrality $c_v$ : 그래프에서 노드가 얼마나 중요한지 고려
- Different ways to model importance
  - Eigenvector centrality
  - Betweenness centrality

- Closeness centrality
- and many others..

### 2.1 Eigenvector centrality

- **Eigenvector centrality:**
  - A node $v$ is important if surrounded by important neighboring nodes $u \in N(v)$.
  - We model the centrality of node $v$ as the sum of the centrality of neighboring nodes:

$$c_v = \frac{1}{\lambda} \sum_{u \in N(v)} c_u$$

$\lambda$ is normalization constant (it will turn out to be the largest eigenvalue of A)

  - Notice that the above equation models centrality in a recursive manner. **How do we solve it?**

Q) 여기서 centrality는 무엇인가(어떤 값인가)? 결정되지 않은 값인가?

A) Eigenvalue-eigenvector 방정식을 푼다 = 고유값 분해를 통해서 고유값 $\lambda$와 고유벡터 $v$를 동시에 구해야하는 연립방정식인 “eigenvalue problem”을 해결하는 것이므로 →$\lambda$와 $v$가 동시에 결정됨.

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

$\lambda$ is some positive constant

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

- node v는 중요한 이웃 node들로 둘러쌓여 있을 때 그 중요성이 크다할 수 있다.

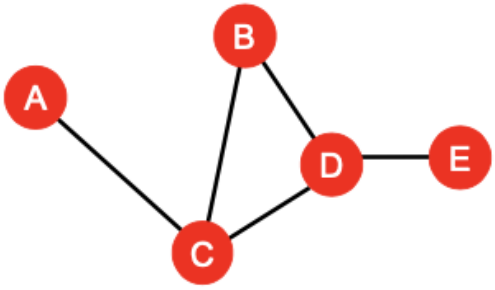
- 따라서 node v의 centrality를 이웃 노드들의 centrality의 합으로 정의한다.
- 위 식은 **recursive equation이며 matrix 형태로 변환**할 수 있다.
- *A*는 node들 간의 관계를 나타내는 인접행렬이며 *c*는 Centrality vector이므로 *c*가 eigenvector임을 알 수 있다.
- The eigenvector $C_{max}$ corresponding to $\lambda_{max}$ *is used for centrality.*

### 2.2 Betweenness centrality

- A node is important if it lies on many shortest paths between other nodes.

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

- **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)

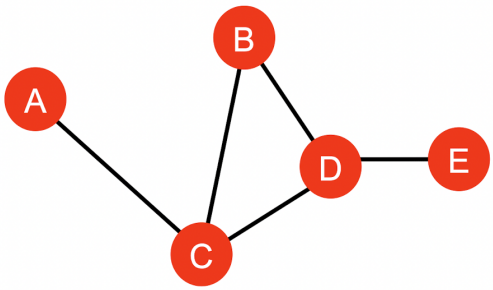
- 어떤 node가 다른 node들의 shortest path를 연결해주는 징검다리 역할을 한다면 중요성이 크다고 할 수 있다.
- node s와 t를 잇는 모든 shortest path중 node v를 거치는 경우의 비율을 c_v로 정의한다.

### 2.3 Closeness centrality

- A node is important if it has small shortest path lengths to all other nodes.

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

- **Example:**




$c_A = 1/(2+1+2+3) = 1/8$
(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)

- 다른 node들과의 거리가 짧을 경우 그 node는 중요성이 크다고 볼 수 있다.
- 모든 node로 가는 shortest path의 길이의 역수로 c_v를 정의한다.

3. **Clustering Coefficient**

- Measures how connected ***v***'s neighboring nodes are:

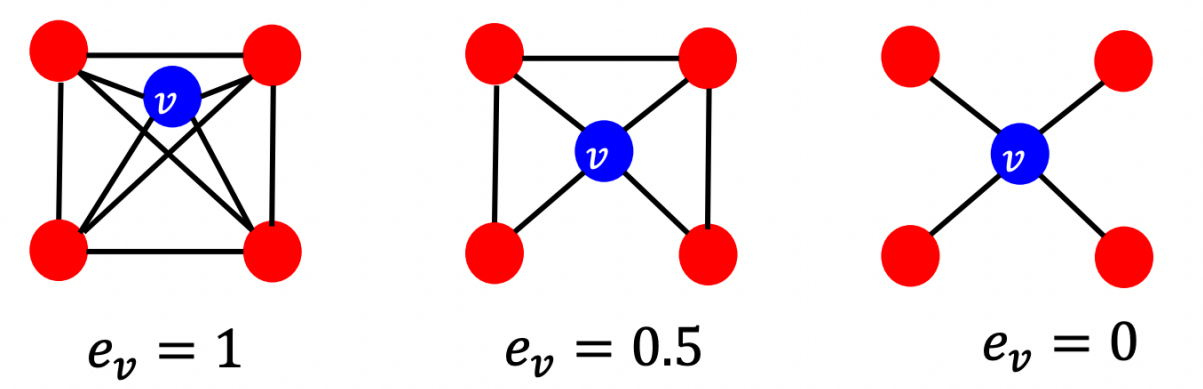
$$e_v = \frac{\#(\text{edges among neighboring nodes})}{\binom{k_v}{2}} \in [0,1]$$

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

—> 연결 가능한 노드 중에서 실제로 몇개가 연결되어 있는가?

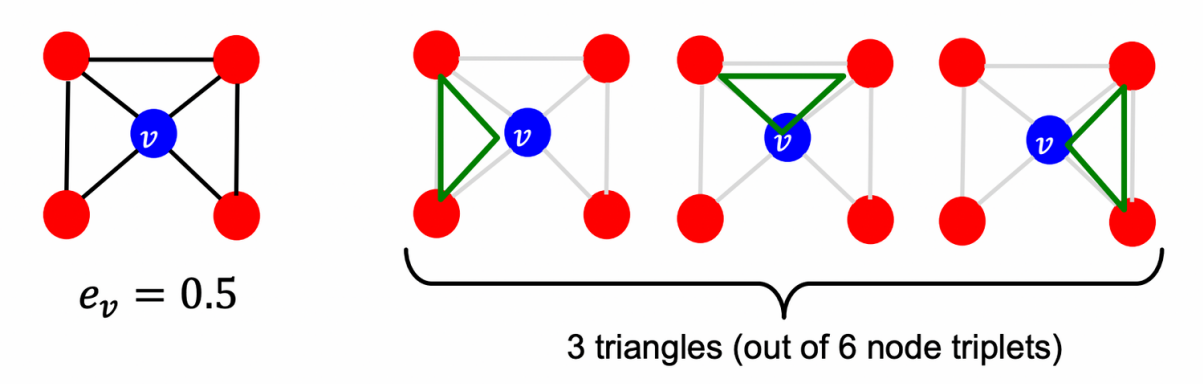
0 : 하나도 연결 x, 1 : 연결 가능한 모든 노드를 연결

- 이웃 node들 간의 연결성을 나타내는 계수로 그 값이 높을수록 clustering되어 있다고 볼 수 있음

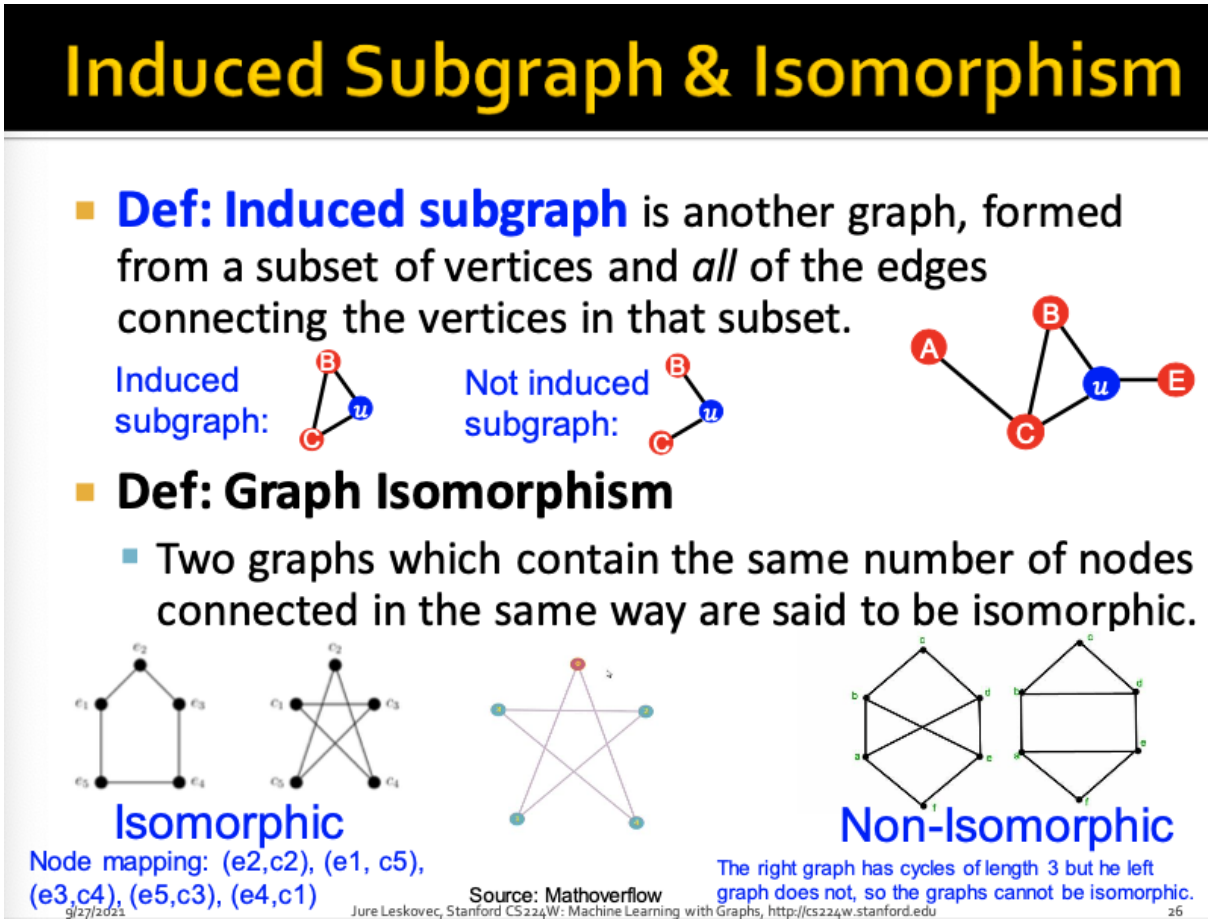



💡 **Observation** : Clustering coefficient counts the #(triangles) in the ego-network

ego-network = 노드 *v* 와 그 주변 노드를 의미 = neighborhood network around a given node

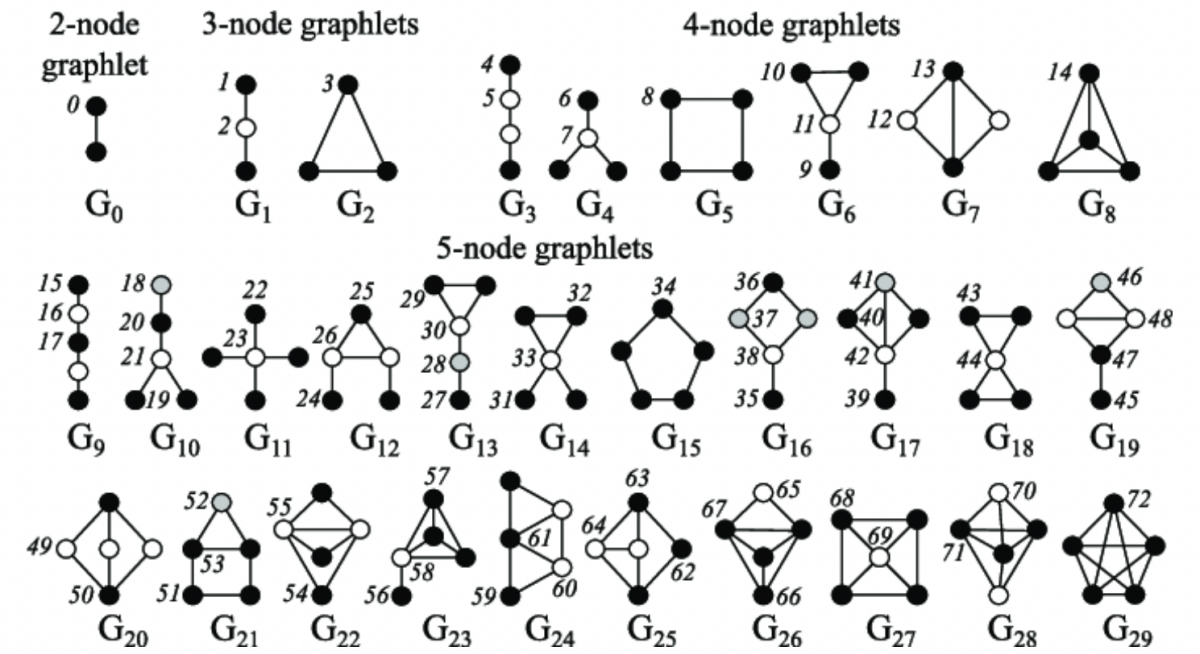



💡 We can generalize the above by counting #(pre-specifiec subgraphs, i.e., **graphlets**).




4. **Graphlets**

- **Goal : Describe network structure around node u**
  - **Graphlets are small subgraphs that describe the structure of node *u*'s network neighborhood**
- **Rooted connected induced non-isomorphic subgraphs:**



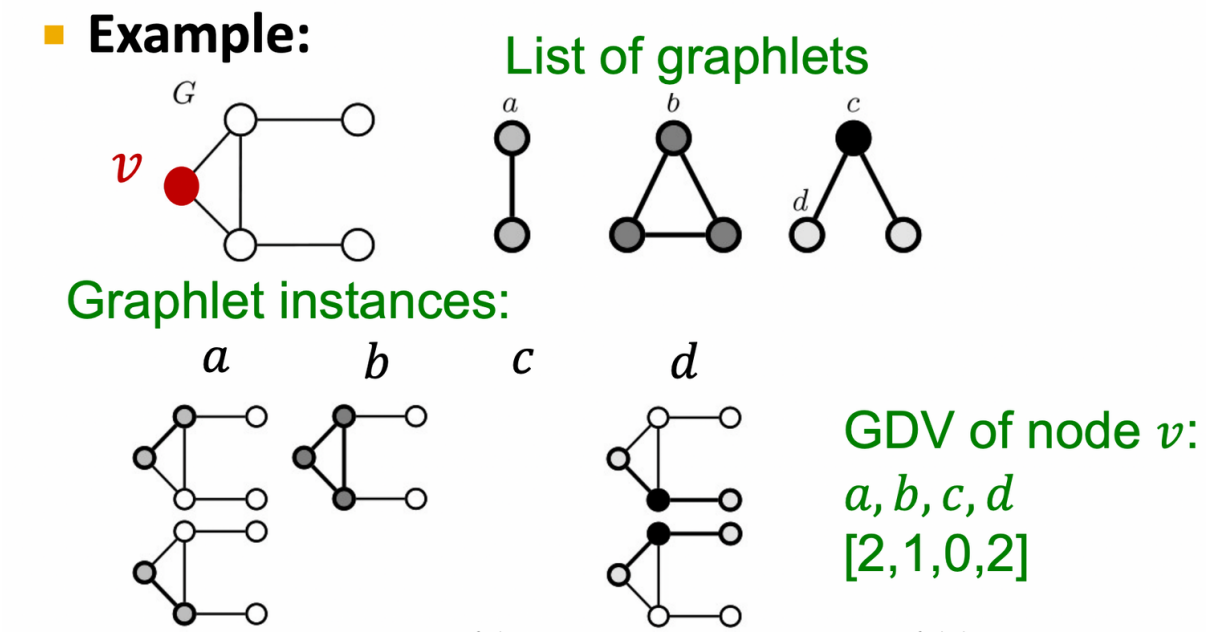

+) 3-node graphlets에서 1번 노드 =/= 2번 노드 지만, 1번 노드는 2번 아래에 있는 노드와는 (회전하면) 같은 노드로 볼 수 있다. → 1번과 2번은 *non-isomorphic*, 1번과 2번 아래의 노드는 *isomorphic*하다고 할 수 있다.

Considering graphlets on **2 to 5** (노드 2개짜리 ~ 5개짜리 graphlet) nodes we get :

- **Vector of 73 coordinates** is a signature of a node that describes the topology of node's neighborhood → 총 73개의 서로 다른 graphlet이 존재할 수 있음 → graphlet이 노드 이웃의 위상 구조 설명
- Captures its interconnectivities out to a distance of **4 hops** → 5개짜리 graphlet은 노드 *v*로부터 4hop 떨어진 (length = 4) path가 몇개 있는지 count하는 것!
- 두개 노드를 가지고 비교하는 node degrees or clustering coefficient 보다 **더 자세한 local topological similarity를 측정 가능!**

**Graphlet Degree Vector(GDV)**

: A count vector of graphlets rooted at a given node




- **non-isomorphic subgraphs로 주어진 node를 특징화** 할 수 있다.
- **Graphlet Degree Vector(GDV)**는 node의 지역적 위상을 측정해주는 척도로 두 node의 비교는 **그 node의 주변이 얼마나 비슷한지**를 나타낸다.
- node *v*를 root로 할 때 *a*,*b*,*c*,*d*가 root인 subgraph와 동일한 모양을 가지는 경우의 수를 벡터 형태로 담아 GDV를 나타낸다.

## Summary

- **We have introduced different ways to obtain node features**
- **They can be categorized as :**
  - Importance-based features:
    - Node degree
    - Differend node centrality measures
  - Structured-based features:
    - Node degree
    - Clustering coefficient
    - Graphlet degree vector

- **Importance-based features**: capture the importance of a node in a graph
  - Node degree:
    - Simply counts the number of neighboring nodes
  - Node centrality:
    - Models importance of neighboring nodes in a graph
    - Different modeling choices: eigenvector centrality, betweenness centrality, closeness centrality
- Useful for predicting influential nodes in a graph
  - **Example:** predicting celebrity users in a social network

- **Structure-based features**: Capture topological properties of local neighborhood around a node.
  - **Node degree:**
    - Counts the number of neighboring nodes
  - **Clustering coefficient:**
    - Measures how connected neighboring nodes are
  - **Graphlet degree vector:**
    - Counts the occurrences of different graphlets
- **Useful for predicting a particular role a node plays in a graph:**
  - **Example:** Predicting protein functionality in a protein-protein interaction network.

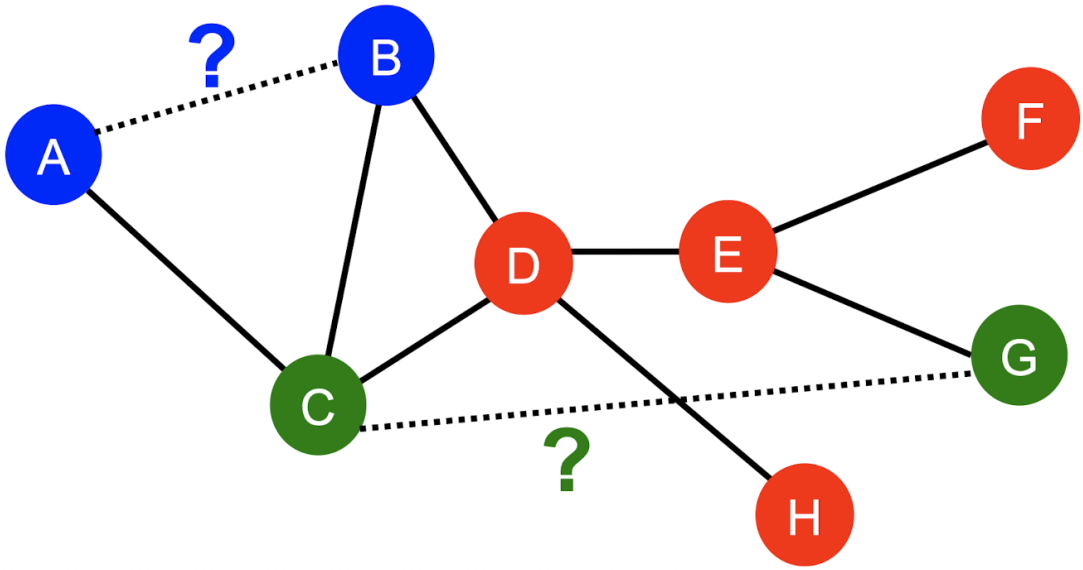
---

## 2. Traditional Link Prediction Task and Features

**Recap**

- The task is to predict **new links** based on the existing links.

- At test time, node pairs(with no existing links) are ranked, and top *K* node pairs are predicted.
- **The key is to design features for a pair of nodes.**




단순히 노드 pair의 feature를 concat해서 train하는 방법은 좋지 X → 두 노드 사이 관계에서 중요한 정보를 잃을 수 있음.

Two formulations of the link prediction task:

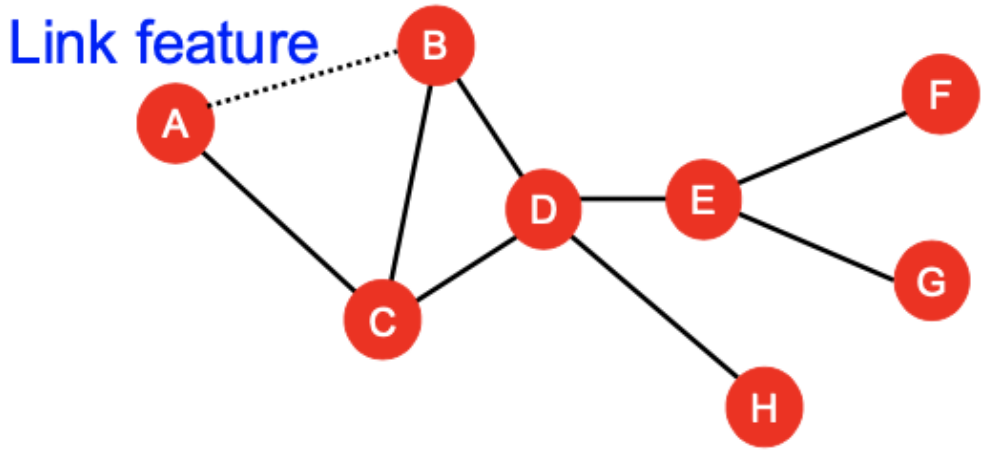
1. **Links missing at random: 랜덤하게 제거한 링크를 예측**
   - Remove a random set of links and then aim to predict them
   - static network 예측에 유리
     - ex) protein-protein interaction network
2. **Links over time: 시간에 따라 예측**
   - Given : $G[t0,t'0]$ a graph on edges up to time $t'0$ ($t0$ 와 $t'0$ 사이에 발생한 노드)
   - output : a **ranked list L** of links (not in $G[t0,t'0]$) that are predicted to appear in $G[t1,t'1]$ (다음 step $t1$ ~ $t'1$ 에 발생할 노드의 랭킹을 예측)
   - Evaluation (학습 방법) :
     - $n$= | *Enew* | : # new edges that appear during the test period $[t1,t'1]$
     - L에서 top n 개의 edge를 예측 → 실제로 발생한 edge와 비교하여 count correct edges
   - 시간에 따라 evolve하는 네트워크 예측에 유리

- ex) social media network, citation network ...

- **Evaluation:**
  - $n = |E_{new}|$: # new edges that appear during the test period $[t_1, t_1']$
  - Take top $n$ elements of $L$ and count correct edges

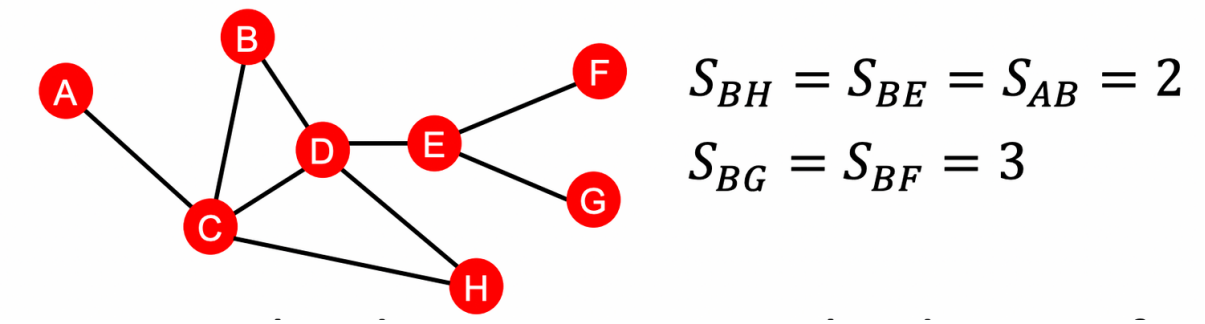
## Link-Level Features : Overview

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




## 2.1 Distance-Based Features

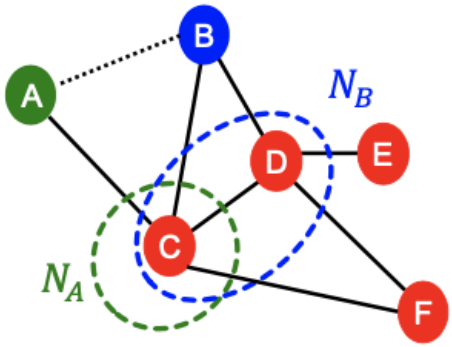
- **Shortest-path distance between two nodes**



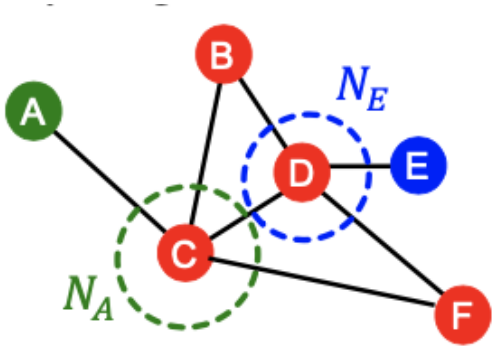

- However, this does not capture the degree of neighborhood overlap:
  - 예를 들어, Node pair (B, H) 는 B→ D → H , B → C → H 두 종류의 최단경로를 가짐
  - pairs (B, E) and (A, B) 같은 경우에는 1 개의 최단경로를 가짐
  - 하지만 이 둘을 똑같이 취급함. ⇒ Strength of connection를 capture하기 위해서 2.2가 등장

## 2.2 Local Neighborhood Overlap

- Captures # neighboring nodes shared between two nodes v1 and v2 :
  - **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: $\frac{1}{\log(k_C)} = \frac{1}{\log 4}$




- Limitation of local neighborhood features:
  - Metric is always zero if the two nodes do not have any neighbors in common
  - However, the two nodes may still potentially be connected in the future.
  - 이러한 한계를 보완하기 위해 2.3이 등장




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

- **Global neighborhood overlap** metrics resolve the limitation by considering the entire graph

## 2.3 Global Neighborhood Overlap

- **Katz index** : count the number of walks of all lengths between a given pair of nodes.

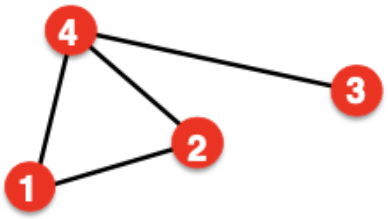
💡 How to compute #paths between two nodes?

⇒ Use **powers** of the `graph adjacency matrix`!

- Computing #walks between two nodes

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

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




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

- **How to compute $P_{uv}^{(2)}$ ?**
  - **Step 1:** Compute **#walks** of length 1 **between each of $u$'s neighbor and $v$**
  - **Step 2**: **Sum up** these #walks across u's neighbors
  - $P_{uv}^{(2)} = \sum_i A_{ui} * P_{iv}^{(1)} = \sum_i A_{ui} * A_{iv} = A_{uv}^2$

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

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

- How to compute #walks between two nodes?
- Use **adjacency matrix powers**!
  - $A_{uv}$ specifies #walks of length 1 (direct neighborhood) between $u$ and $v$.
  - $A^2_{uv}$ specifies #walks of **length 2** (neighbor of neighbor) between $u$ and $v$.
  - And, $A^l_{uv}$ specifies #walks of **length $l$**.

- **Katz index** between $v_1$ and $v_2$ is calculated as

**Sum over *all walk lengths***

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

$A^l_{v_1 v_2}$: #walks of length $l$ between $v_1$ and $v_2$

$0 < \beta < 1$: discount factor

- Katz index matrix is computed in closed-form:

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

- discount factor $\beta$ = length에 따라 exponentially 감소하면서 longer path에 더 낮은 importance를 부여하게 된다.

## Summary

- Distance-based features:
  - Uses the shortest path length between two nodes but does not capture how neighborhood overlaps.
- Local neighborhood overlap:

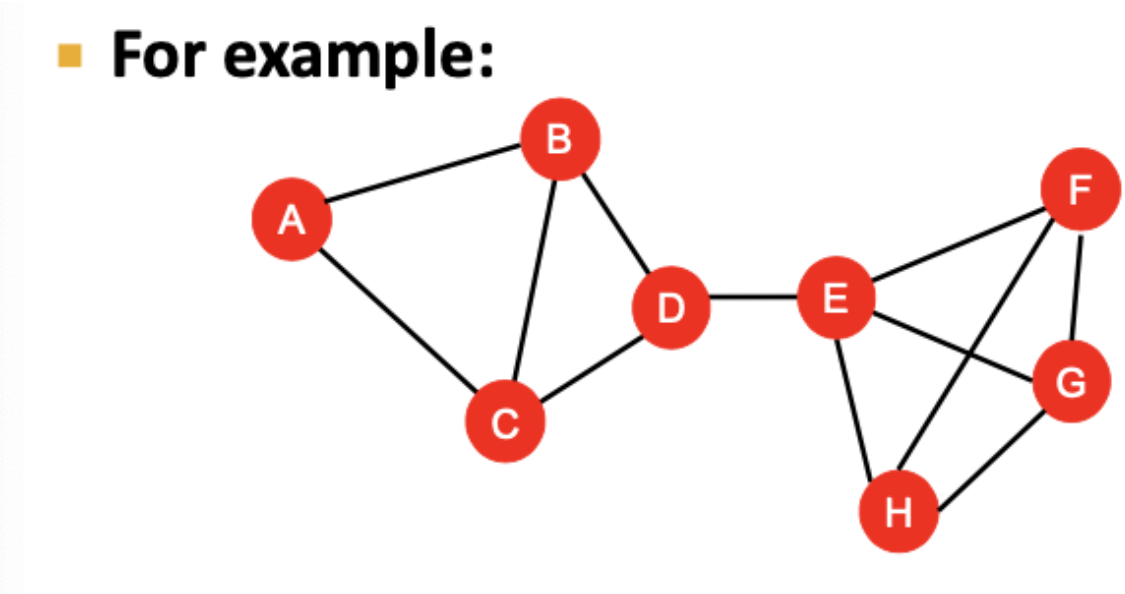
    - Captrues how many neighboring nodes are shared by two nodes.
    - Becomes zero when no neighbor nodes are shared.
- Global neighborhood overlap:
    - Uses global graph structure to score two nodes.
    - Katz index counts #walks of all lengths between two nodes.

---

# 3. Traditional Graph-Level Features and Graph Kernels

## Graph-Level Features

- **Goal** : We want features that characterize the structure of an entire graph.




## Background : Kernel Methods

- **Idea : Design Kernels instead of feature vectors.**
- **A quick introduction to Kernels:**

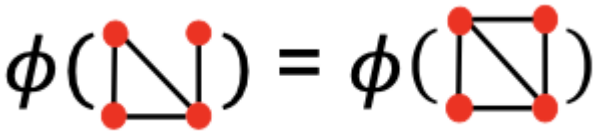
- Kernel $K(G, G') \in \mathbb{R}$ measures similarity b/w data
- Kernel matrix $\boldsymbol{K} = \left(K(G, G')\right)_{G,G'}$, must always be positive semidefinite (i.e., has positive eigenvalues)
- There exists a feature representation $\phi(\cdot)$ such that $K(G, G') = \phi(\mathrm{G})^{\mathrm{T}}\phi(G')$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

## Graph-Level Features : Overview

- Graph Kernels : Measure similarity between two graphs:
  - Graphlet Kernel
  - Weisfeiler-Lehman Kernel
  - Other kernels are also proposed in the literature(beyond the scope of this lecture)
    - Random-walk kernel
    - Shortest-path graph kernel
    - And many more...

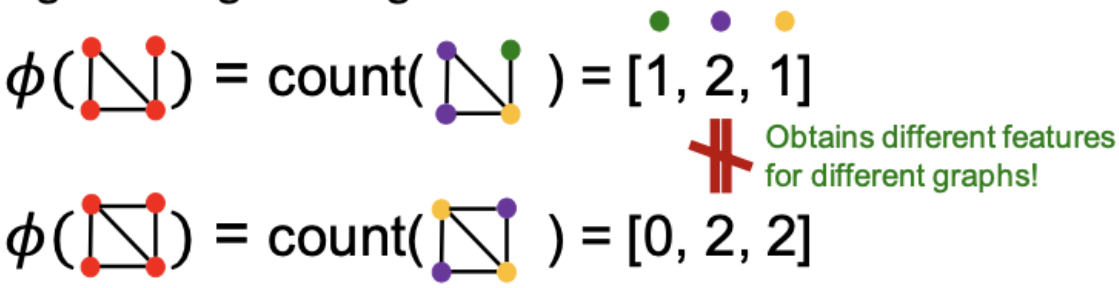
## Graph Kernel : Key Idea

- Goal : Design graph feature vector $\phi(G)$
- Key idea : Bag-of-Words(BOW) for a graph
  - **Recall :** Bow simply uses the word counts as features for documents(no ordering considered).
  - Naive extension to a graph : **Regard nodes as words.**
  - Since both graphs have **4 red nodes**, we get the same feature vector for two different graphs...



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

Deg1: ● Deg2: ● Deg3: ●

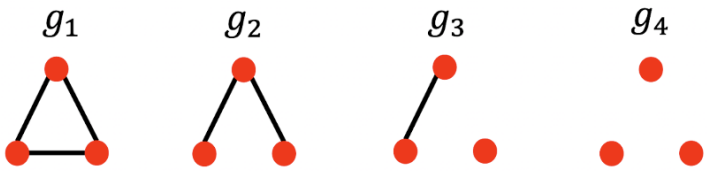



- Both Graphlet Kernel and Weisfeiler-Lehman (WL) Kernel use **Bag-of-*** representation of graph, where * is more sophisticated than node degrees!

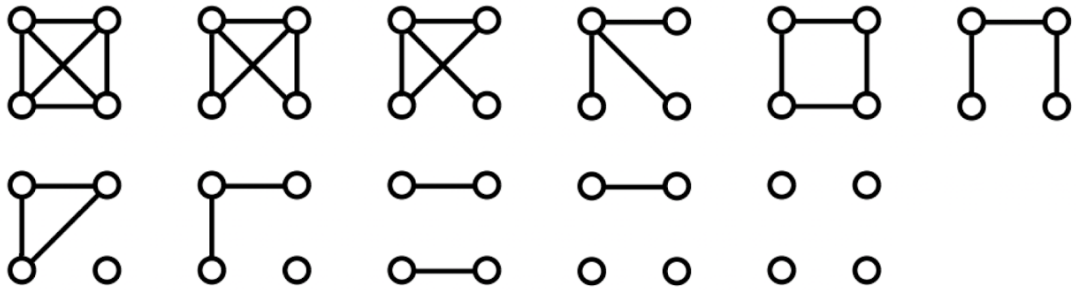
## Graphlet Features

- **Key idea** : Count the number of different graphlets in a graph.
  - 💡 Note: node-level features에서 배운 graphlets 정의와 다르다 !
  - The two differences are:
    - Nodes in graphlets here do **not need to be connected**(allows for isolated nodes)
    - The graphlets here are not rooted.

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



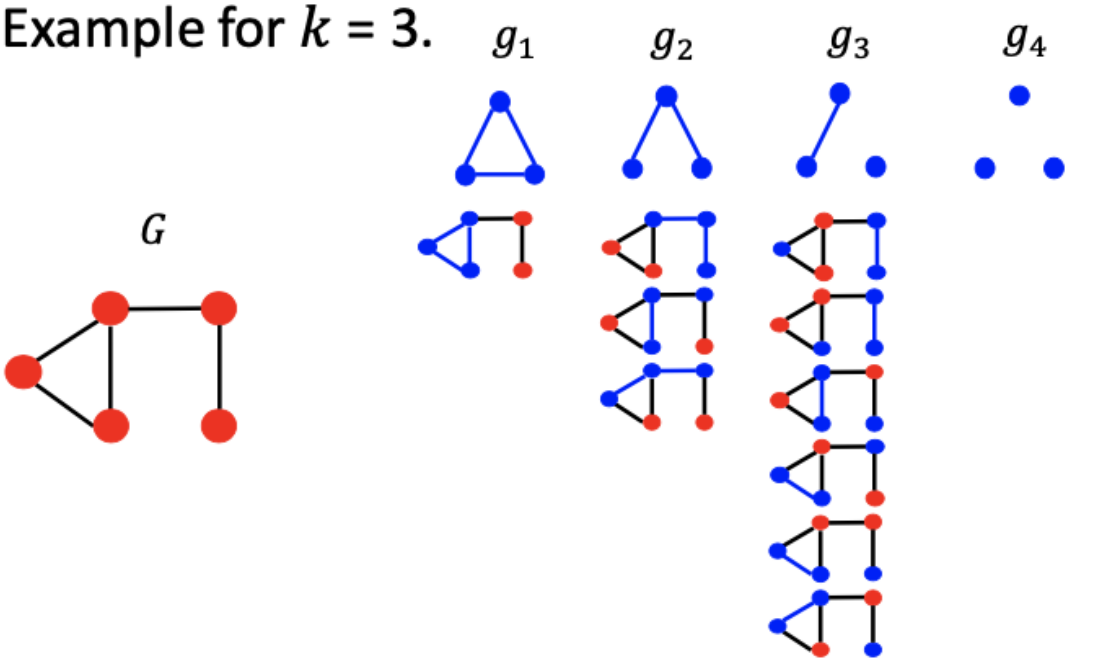

- For $k = 4$ , there are 11 graphlets.



- Given graph $G$, and a graphlet list $\mathcal{G}_k = (g_1, g_2, \ldots, g_{n_k})$, define the graphlet count vector $f_G \in \mathbb{R}^{n_k}$ as

$$(f_G)_i = \#(g_i \subseteq G) \text{ for } i = 1,2,\ldots,n_k.$$

- Example for $k = 3$.




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

- 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

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

**Limitations**: Counting graphlets is expensive!

- Counting size-$k$ graphlets for a graph with size $n$ by enumeration takes $n^k$.
- This is unavoidable in the worst-case since subgraph isomorphism test (judging whether a graph is a subgraph of another graph) is **NP-hard**.
- If a graph's node degree is bounded by $d$, an $O(nd^{k-1})$ algorithm exists to count all the graphlets of size $k$.

**Can we design a more efficient graph kernel?**

### Weisfeiler-Lehman Kernel

- Goal: design an efficient graph feature descriptor $\phi(G)$
- idea: use neighborhood structure to iteratively enrich node vocabulary.
  - Bag of node degrees를 one-hop ~ multi-hop으로 일반화 한 것!
  - Algorithm to achieve this: `Color Refinement`

- **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

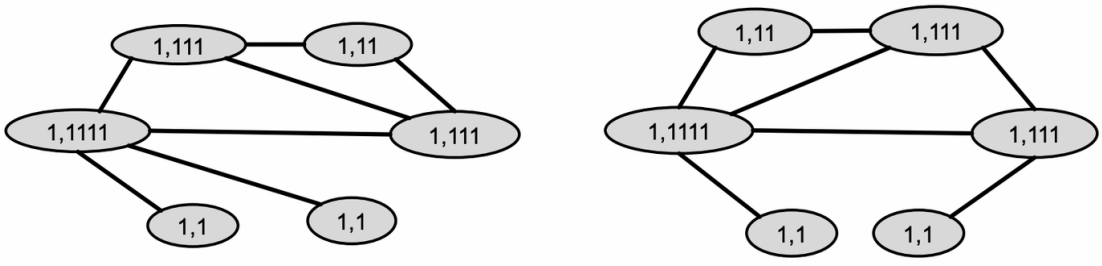
$$c^{(k+1)}(v) = \text{HASH}\left(\left\{c^{(k)}(v), \{c^{(k)}(u)\}_{u \in N(v)}\right\}\right),$$

    where HASH maps different inputs to different colors.
  - After $K$ steps of color refinement, $c^{(K)}(v)$ summarizes the structure of $K$-hop neighborhood

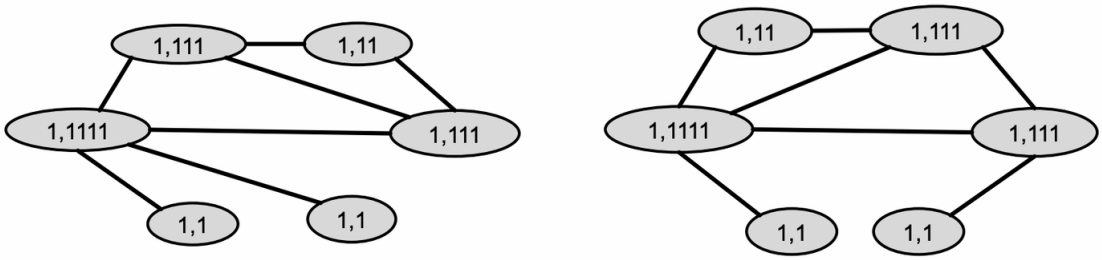
## Example of color refinement given two graphs

1. Assign initial colors


2. Aggregate neighboring colors



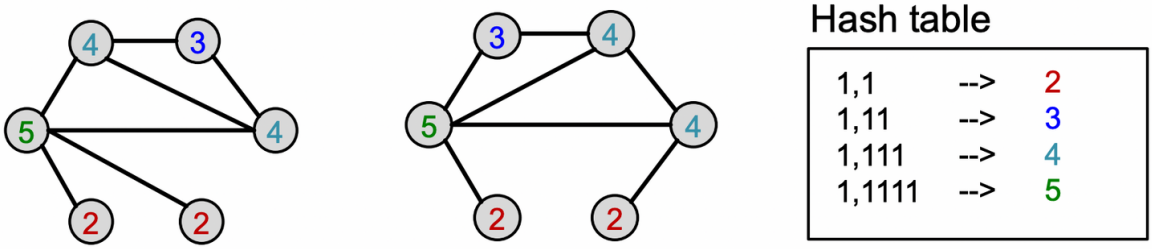

3. Aggregated colors




4. Hash aggregated colors

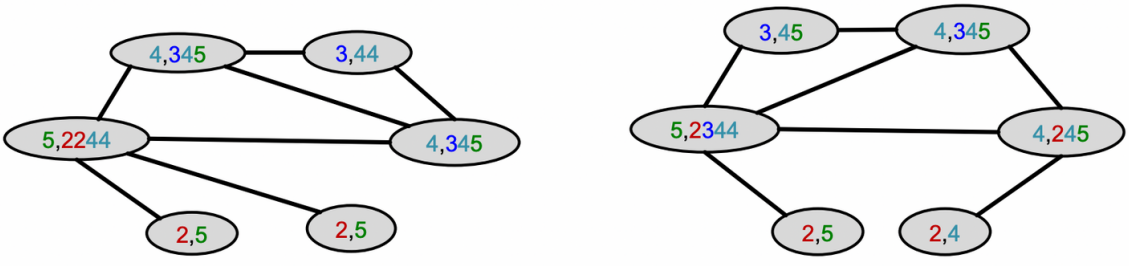
$$c^{(k+1)}(v) = \text{HASH}\left(\left\{c^{(k)}(v), \{c^{(k)}(u)\}_{u \in N(v)}\right\}\right),$$

where HASH maps different inputs to different colors.

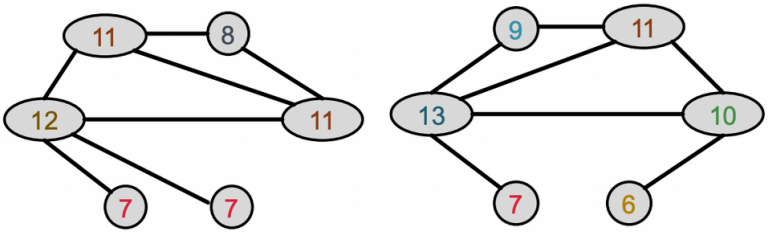



- Aggregated colors


- Hash aggregated colors



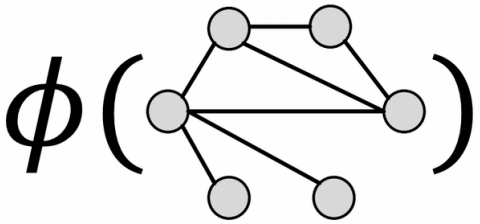
- Hash aggregated colors




Hash table

| | | |
|---|---|---|
| 2,4 | --> | 6 |
| 2,5 | --> | 7 |
| 3,44 | --> | 8 |
| 3,45 | --> | 9 |
| 4,245 | --> | 10 |
| 4,345 | --> | 11 |
| 5,2244 | --> | 12 |
| 5,2344 | --> | 13 |

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

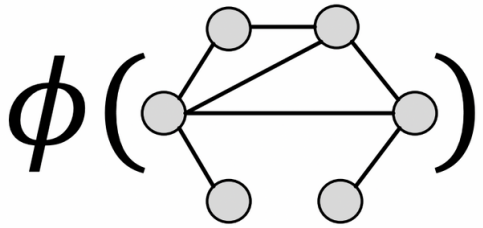


Colors

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

= [6,2,1,2,1,0,2,1,0, 0, ~~0, 2, 1~~]

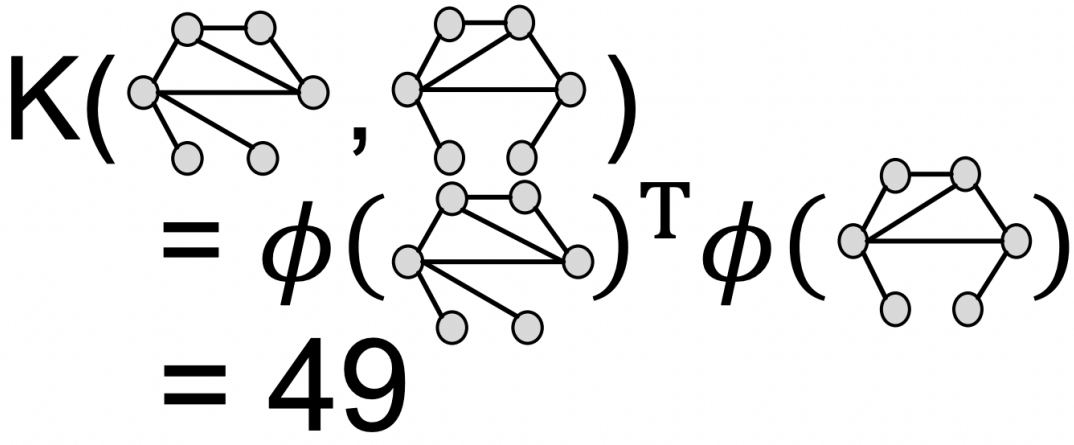
Counts 2 , 1, 0



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

= [6,2,1,2,1,1,1,0,1, 1, 1, 0, 1]

6. The WL kernel value is computed by the inner product of the color count vectors



- WL kernel is **computationally efficient**
  - The time complexity for color refinement at each step is linear in #(edges), since it involves aggregating neighboring colors.
- When computing a kernel value, only colors appeared in the two graphs need to be tracked.
  - Thus, #(colors) is at most the total number of nodes.
- Counting colors takes linear-time w.r.t #(nodes).
- In total, time complexity is **linear in #(edges).**

### Summary

- Graphlet Kernel
  - Graph is represented as **Bag-of-graphlets.**
  - **Computationally expensive**
- Weisfeiler-Lehman Kernel
  - Apply $K$-step color refinement algorithm to enrich node colors
    - Different colors capture different $K$-hop neighborhood structures
  - Graph is represented as **Bag-of-Colors**
  - **Computationally efficient**
  - Closely related to Graph Neural Networks

---

## Today's Summary

- **Traditional ML Pipeline**
  - Hand-crafted feature + ML model
- **Hand-crafted features for graph data**
  - Node-level:
    - Node degree, Node centrality, clustering coefficient, graphlets
  - Link-level:

        - Distance-based feature
        - local/global neighborhood overlap
    - Graph-level:
        - Graphlet kernel, WL kernel

---

# References

https://velog.io/@helloimyj/CS224W-Lec02

https://velog.io/@kimkj38/CS224W-Lecture2-1-Traditional-Feature-based-Methods-Node