2️⃣

Traditional Methods for Machine Learning in Graphs

Traditional ML Pipelin

- 예측의 범위는 node, link, graph level로, 각 범위에 따른 feature을 적절히 디자인
- 고려해야 할 feature은 두 가지로, structural feature과 nodes' attribute

  → 이번 강의에서는 structural feature을 다룬다
- 파이프라인

  step1. node / link / graph 를 feature vector로 표현한 후 훈련

  step2. 훈련된 모델을 적용해 예측

이번 Lecture의 주된 목표 : Feature Design

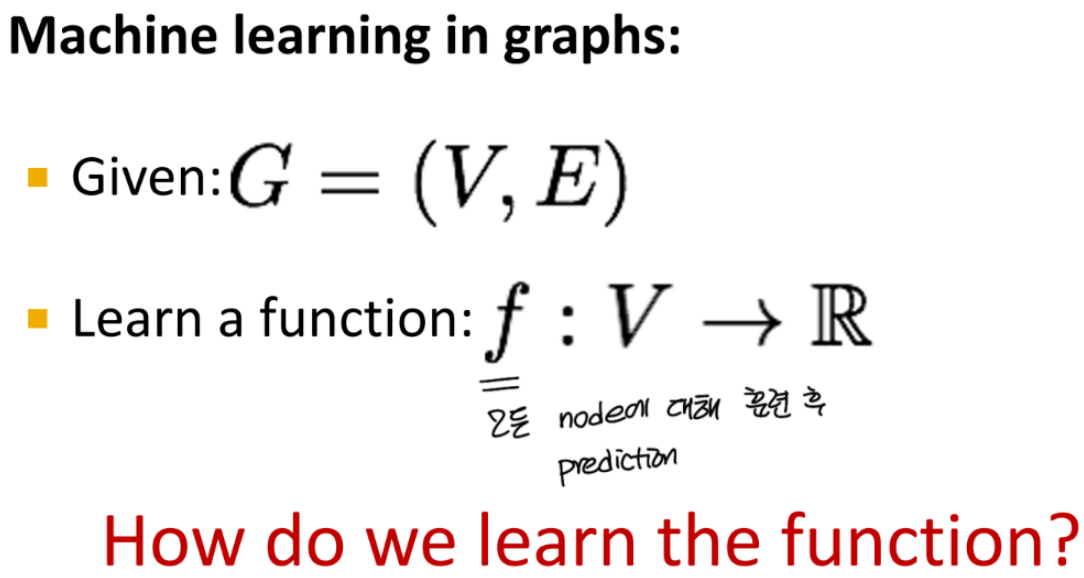
- 그래프의 effective feature들을 사용하는 것이 모델 성능의 향상에 key이다
- traditional ML pipeline들은 hand-designed feature을 사용
  - 이번 lecture에서는 다음과 같은 traditional feature을 다룬다
  - node / link / graph - level prediction
- 이때 simplicity를 위해 undirected graph들을 대상으로 함

Machine Learning in Graphs

- 목표 : object 집합에 대한 prediction을 하는 것
- Design choices
  - feature : d 차원의 vectors
  - objects : nodes, edges, set of nodes, entire graphs

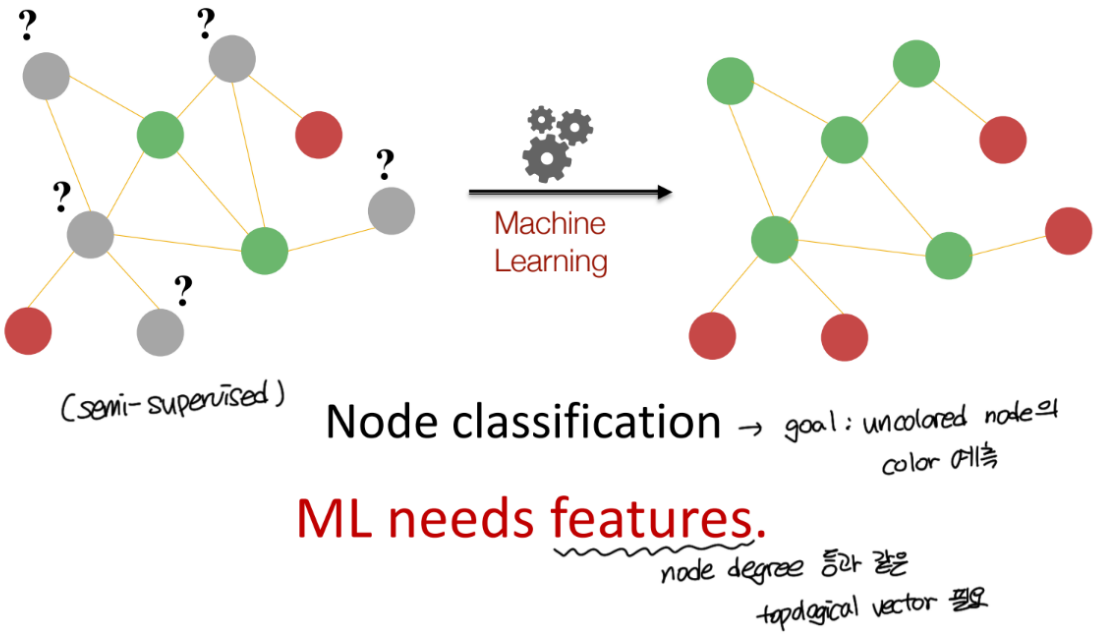
- object function : 어떤 일을 수행하는지

- graph



Node-level Tasks and Features

- Node-level tasks
  - node classification : graph, 즉 topological vector,을 이용해 node의 color을 예측
  - 이때 graph의 노드들은 서로 연결(관련)되어 있어 기존보다 dependency에 초점을 맞춰야 함



- Node-level features
  - 네트워크에서 노드의 구조와 위치를 characterize
  - 종류
    - node degree
    - node centrality
    - clustering coefficient
    - graphlets

- node의 중요성에 초점을 맞춘 feature
  - node degree
    - 해당 node가 갖고 있는 edge 개수
    - 문제점 : 주변 노드들을 공통적으로 treat해, degree가 같으면 ML model들은 각기 다른 노드들에 대해 똑같은 value로 predict함
  - node centrality
    - network 안에서 node가 얼마나 중요한가 → network 중심과 얼마나 가까운가
    - 계산 방법
      1. eigenvector centrality

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

λ is some positive constant (normalizing vector)

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

- We see that centrality is the eigenvector! (eigenvector that is associated with the largest eigenvalue)
- The largest eigenvalue λ_{max} is always positive and unique (by Perron-Frobenius Theorem).
- The leading eigenvector $\boldsymbol{c}_{max}$ is used for centrality.

2. betweenness centrality

   해당 노드가 transit hub로서 얼마나 중요한가

- **Betweenness centrality:** ("해당 node가 얼마나 transit hub로서 중요한가")
  - A node is important if it lies on many shortest paths between other nodes. (node가 중간에 bridge라면 high importance)

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

3. closeness centrality

   특정 노드와 다른 모든 노드 간 거리가 얼마나 가까운가

- **Closeness centrality:** ("position of node")
  - A node is important if it has small shortest path lengths to all other nodes. (center에 가까울수록 모든 node에 대해 짧은 path를 지난다)

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

↳ center에서 멀어질수록 c_v는 낮아짐

- 해당 노드가 어떤 구조를 갖고 있는지에 초점을 맞춘 feature
  - clustering coefficient

- 특정 노드와 이웃하는 노드들이 얼마나 연결되어 있는가 (local structure에 초점)

  → 주변에 얼마나 potential edge들이 많은가

node degree와 local structure around the node의 focus

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

how many actually occurs

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

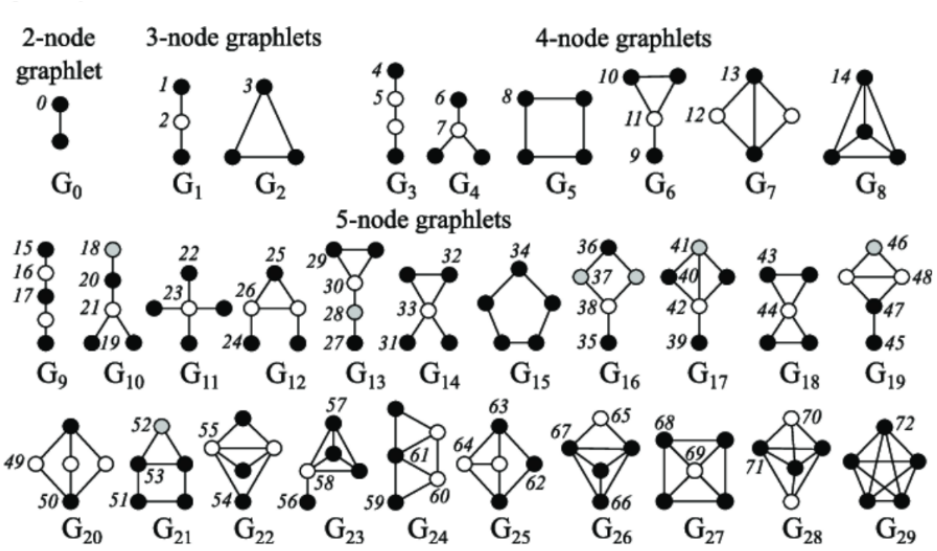
all connected

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

- graphlets
  - clustering coefficient는 ego-entwork에서 triangle의 수를 센다

    → graphlet에서는 이걸 일반화 by graphlet counting

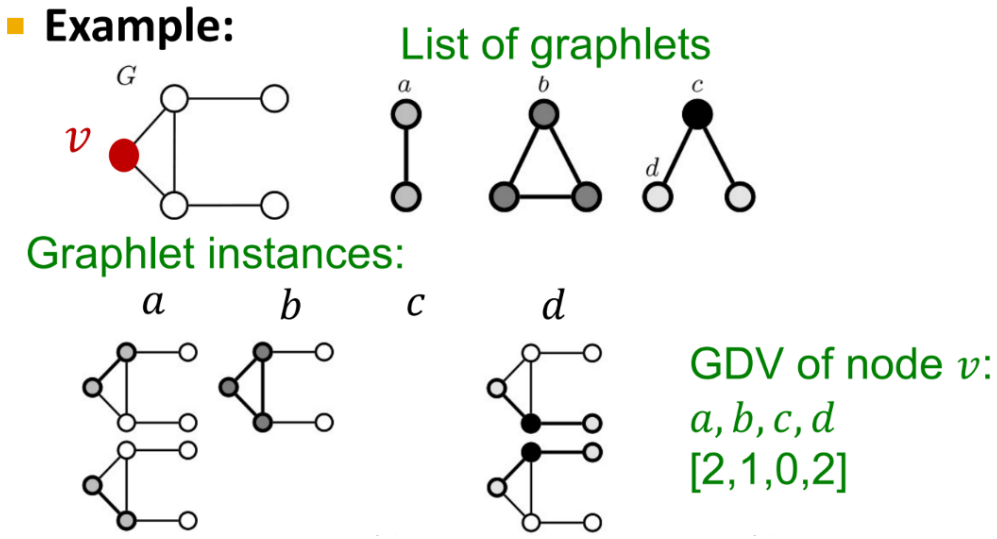
  - graphet이란 rooted connected non-isomorphic subgraph들이다



  - Graphlet Degree Vector (GDV) : 노드들과 연결된 graphlet 수

    → node의 local network topology를 알 수 있게 해준다

    (예시)



Link Prediction Task and Features

- 목표 : 이미 존재하는 link들을 바탕으로 새로운 link를 예측하는 것

  → 이를 위해 node간 feature을 design해야 하는데, 이때 많은 정보가 손실된다는 단점 존재

- 두 가지 방법
  - link들을 랜덤하게 missing한 후 prediction 진행 → static network에 유용
  - 시간의 흐름에 따라 예측 → naturally evolve하는 network에 유용

- Link-level features
  - 목표 : 두 노드 간 relationship을 describe해 다른 node 간 link를 예측
  - 종류
    1. Distance-based features

       두 노드 간 가장 짧은 거리

       → 이웃 노드들끼리 오버랩되는 정도, 즉 connection strength,를 capture 못 함

    2. Local Neighborhood Overlap

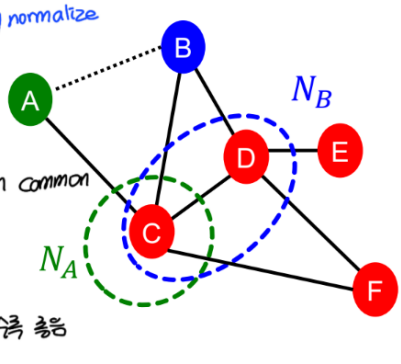
두 노드가 공유하는 이웃 노드들의 수를 capture함

- **Common neighbors:** $|N(v_1) \cap N(v_2)|$ → 높을수록 다른 node들과 neighbor 할 가능성↑
  - Example: $|N(A) \cap N(B)| = |\{C\}| = 1$
- **Jaccard's coefficient:** $\frac{|N(v_1) \cap N(v_2)|}{|N(v_1) \cup N(v_2)|}$ normalize
  - Example: $\frac{|N(A) \cap N(B)|}{|N(A) \cup N(B)|} = \frac{|\{C\}|}{|\{C,D\}|} = \frac{1}{2}$
- **Adamic-Adar index:**

$\sum_{u \in N(v_1) \cap N(v_2)} \frac{1}{\log(k_u)}$ → how many neighbors in common

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

degree가 낮은 neighbor들이 많을수록 높음



3. Global Neighborhood Overlap

서로 공유하는 노드가 없으면 항상 0이 되는 local overlap의 단점을 보완

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

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

Graph-Level Features and Graph Kernels

- 목표 : 전체 graph의 구조를 characterize하는 feature을 디자인
- traditional ML에서 graph-level prediction에는 kernel methods을 사용함

- Graph-level features
  - 두 그래프 간의 similarity 계산
  - 종류
    1. Graphlet Kernel

       하나의 그래프에서 다른 graphlet들의 수를 count (Bag of node degrees를 형성)

       - 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

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

         ↳ proportion(frequency) of given graphlet

    2. Weisfeiler-Lehman Kernel

       computationally expensive한 graphlet kernel의 단점 보완

       color refinement 방법을 이용해 bag of node degrees의 일반화 (hash 함수 이용)

- **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

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

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