CSE8803/CX4803

Machine Learning in Computational Biology

Lecture 14b:
Traditional ML on graphs

Yunan Luo

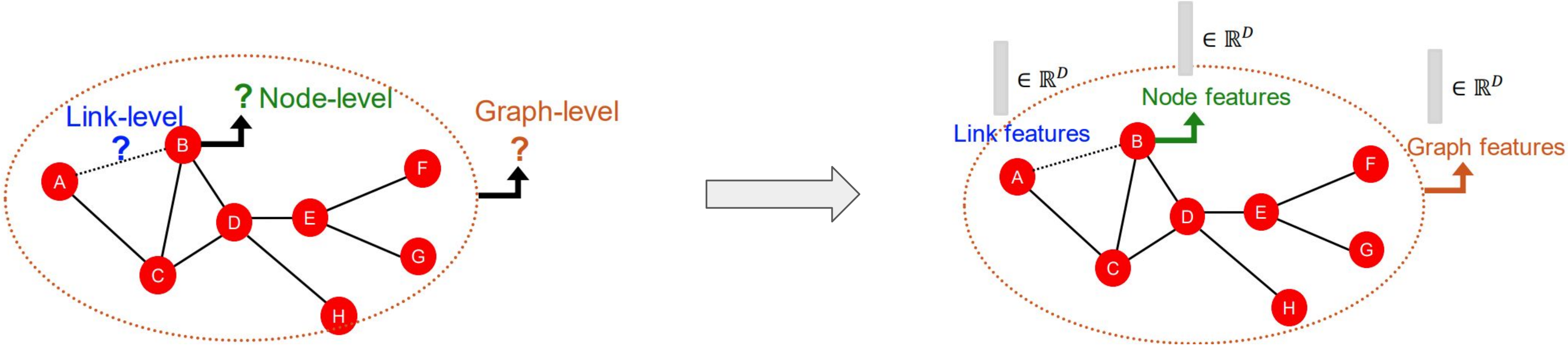
Traditional machine learning for graphs

(for simplicity, we focus on undirected graphs in the lecture)

Machine learning tasks on graphs

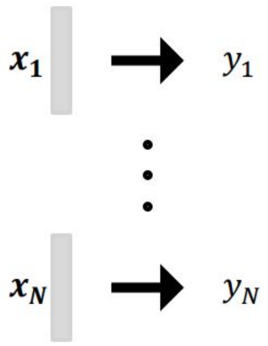
- **Node-level** prediction
- **Link-level** prediction
- **Graph-level** prediction

Design features for
nodes / links / graphs

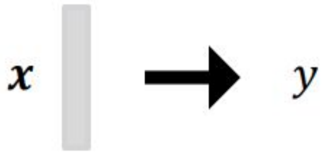


Traditional machine learning pipeline

- **Train an ML model**:
  - Obtain features of training data (node/edge/graph)
  - Train a ML model (SVM, NN, etc)



- **Apply the model**:
  - Given a new node/edge/graph, obtain its features and make a prediction

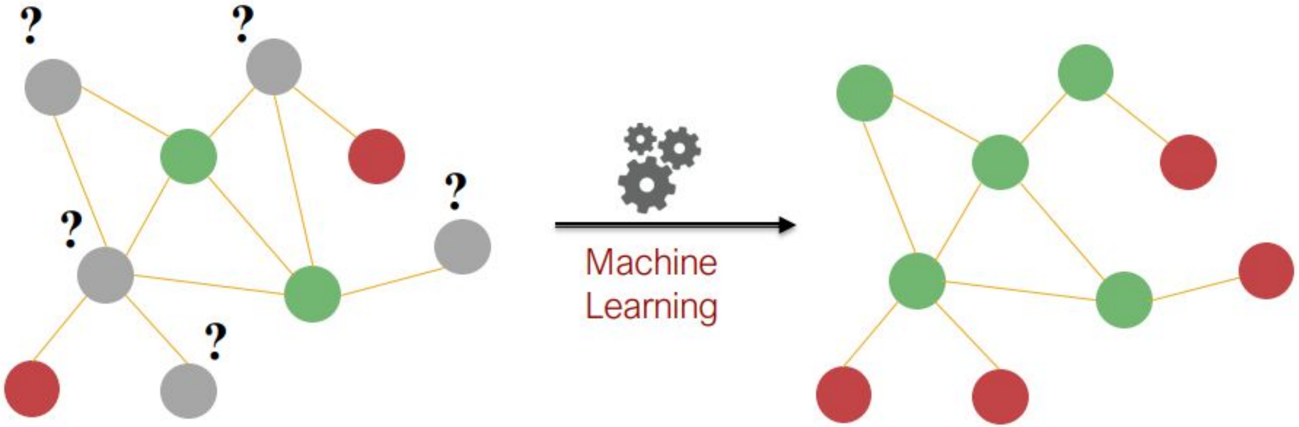


Traditional ML focuses on (manually) designing effective features over graphs

Node-level tasks and features

Node-level tasks

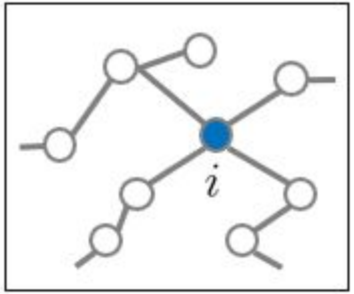
Node classification (e.g., protein function prediction)



Node-level features

Goal: design features that characterize the structure and position of a node in the network

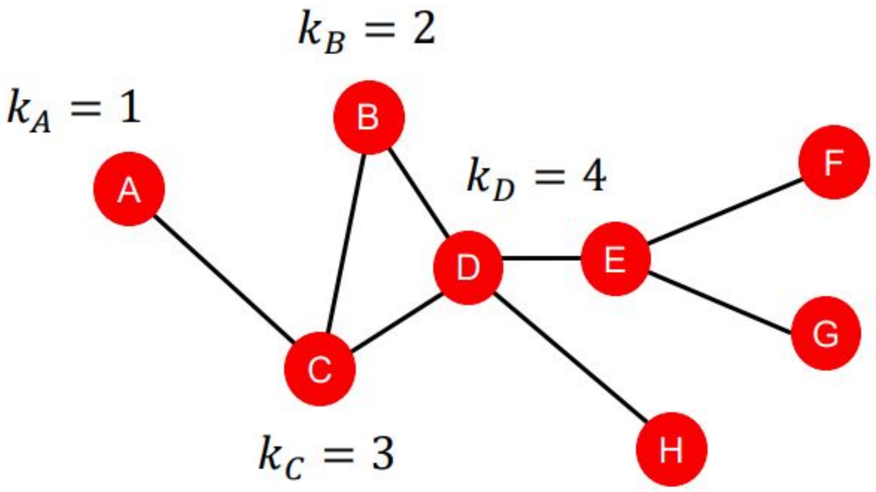
- Node degree
- Node centrality
- Clustering coefficient
- Graphlets



Node degree

Degree $\boldsymbol{k_v}$ of node $\boldsymbol{v}$: the number of edges (neighboring nodes) the nodes has.

- Treats all neighboring nodes equally



Node centrality

- **Node degree** counts the neighboring nodes without capturing their importance
- **Node centrality** $\boldsymbol{c_v}$ takes the node importance in a graph into account
- Different node centrality measures
  - Eigenvector centrality
  - Betweenness centrality
  - Closeness centrality
  - …

Node centrality: eigenvector centrality

- A node v is important if surrounded by important neighboring nodes ***u*** ∈ ***N***(***u***)
- The centrality of a node v is defined as the sum of the centrality of neighboring nodes

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

Recursive definition

λ is a normalization constant

Node centrality: eigenvector centrality

- Rewrite the recursive definition in matrix form

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

λ is a normalization constant

A: adjacency matrix
c: centrality vector
λ: eigenvalue

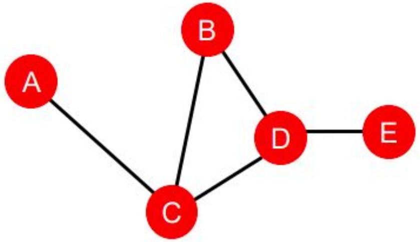
- Centrality ***c*** is the eigenvector of ***A***
- The eigenvector ***c*** corresponding to $\boldsymbol{\lambda_{max}}$ is used for centrality

Node centrality: betweenness centrality

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

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

- **Unnormalized** version:



$$c_A = c_B = c_E = 0$$
$$c_C = 3$$
(A-<u>C</u>-B, A-<u>C</u>-D, A-<u>C</u>-D-E)

$$c_D = 3$$
(A-C-<u>D</u>-E, B-<u>D</u>-E, C-<u>D</u>-E)

- **Normalized** version:
  - Undirected graph
    - Normalized by (N-1)(N-2)/2
  - Directed graph
    - Normalized by (N-1)(N-2)
  - E.g.:

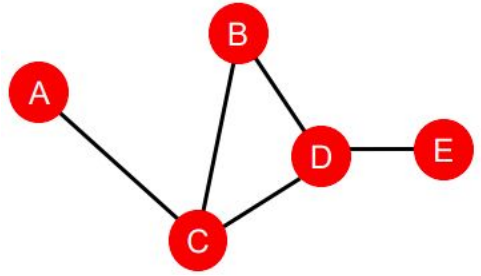
$$c_D = \frac{3}{\binom{N-1}{2}} = 0.5$$

Node centrality: 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)

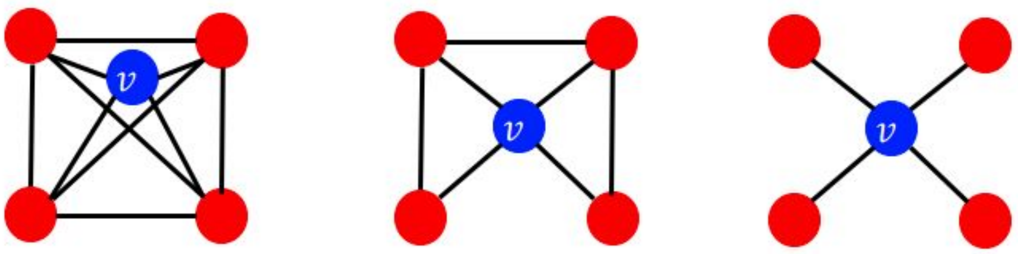
Clustering coefficient

- Measures how connected v's neighboring nodes are

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

Total number of possible edges among ***v***'s neighboring nodes (***k_v***: degree of node ***k***)

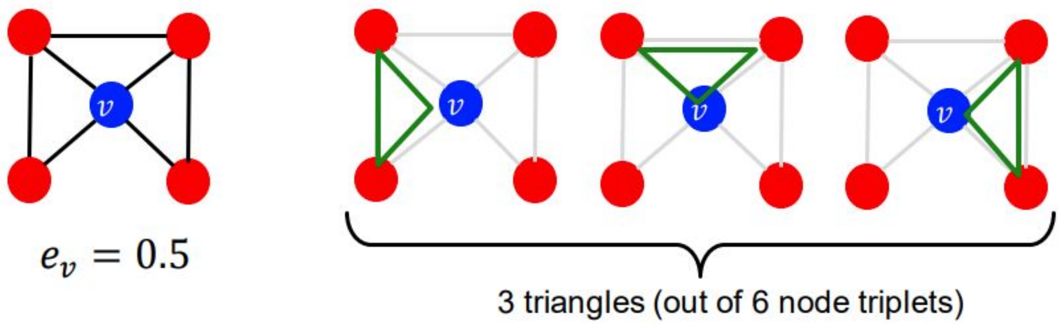
- Example:



$e_v = 1$ $\quad$ $e_v = 0.5$ $\quad$ $e_v = 0$

Node features: graphlets

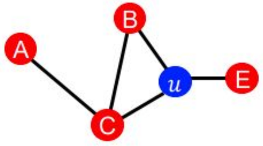
- Clustering coefficient of ***u*** counts the #(▲) node ***u*** touches



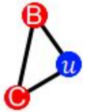
- Can we generalize the counting of ▲ to other pre-specified subgraphs (graphlets)?

Induced subgraph & isomorphism

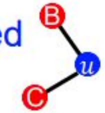
- **Def**: **Induced subgraph** is another graph, formed from a subset of vertices and all of the edges connecting the vertices in that subset.



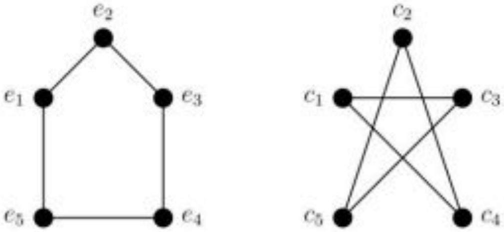
Induced subgraph:



Not induced subgraph:



- **Def**: **Graph Isomorphism**
  - Two graphs which contain the same number of nodes connected in the same way are said to be isomorphic.

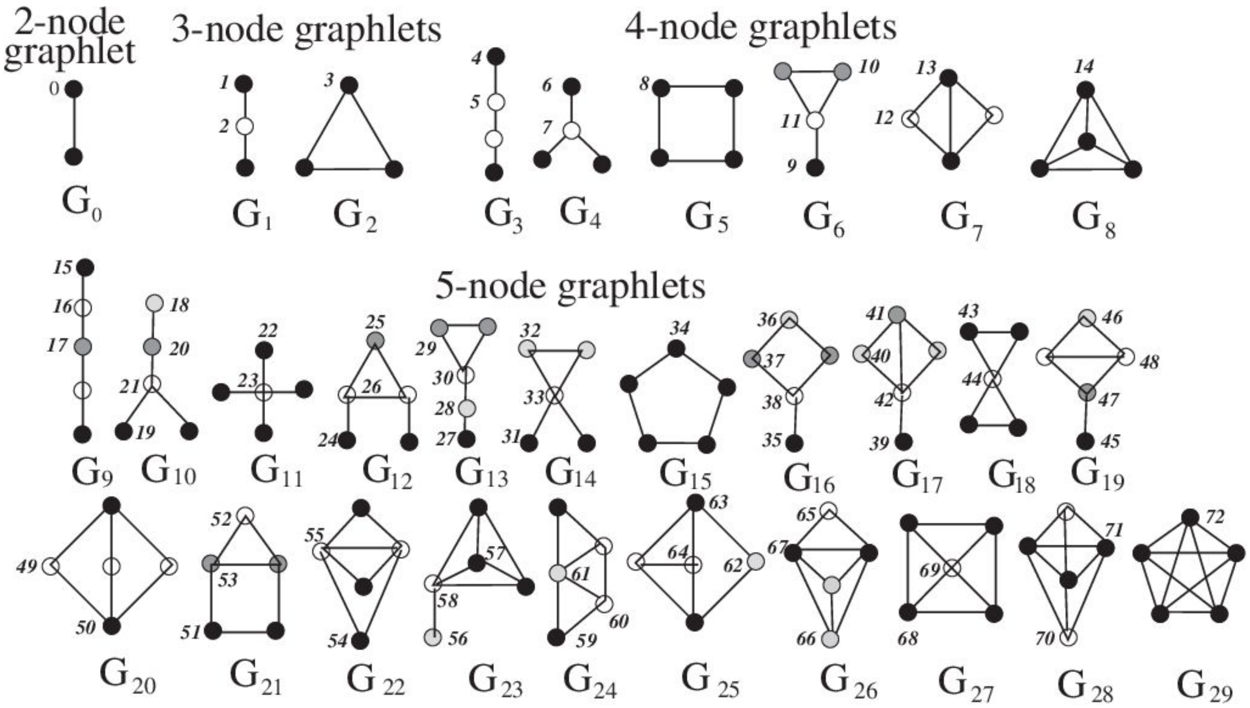


Isomorphic

Node mapping: (e2,c2), (e1, c5), (e3,c4), (e5,c3), (e4,c1)

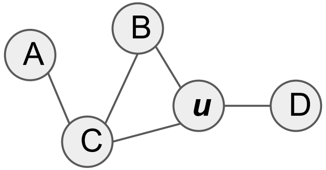
Node features: graphlets

- **Graphlets**: Rooted connected induced non-isomorphic subgraphs

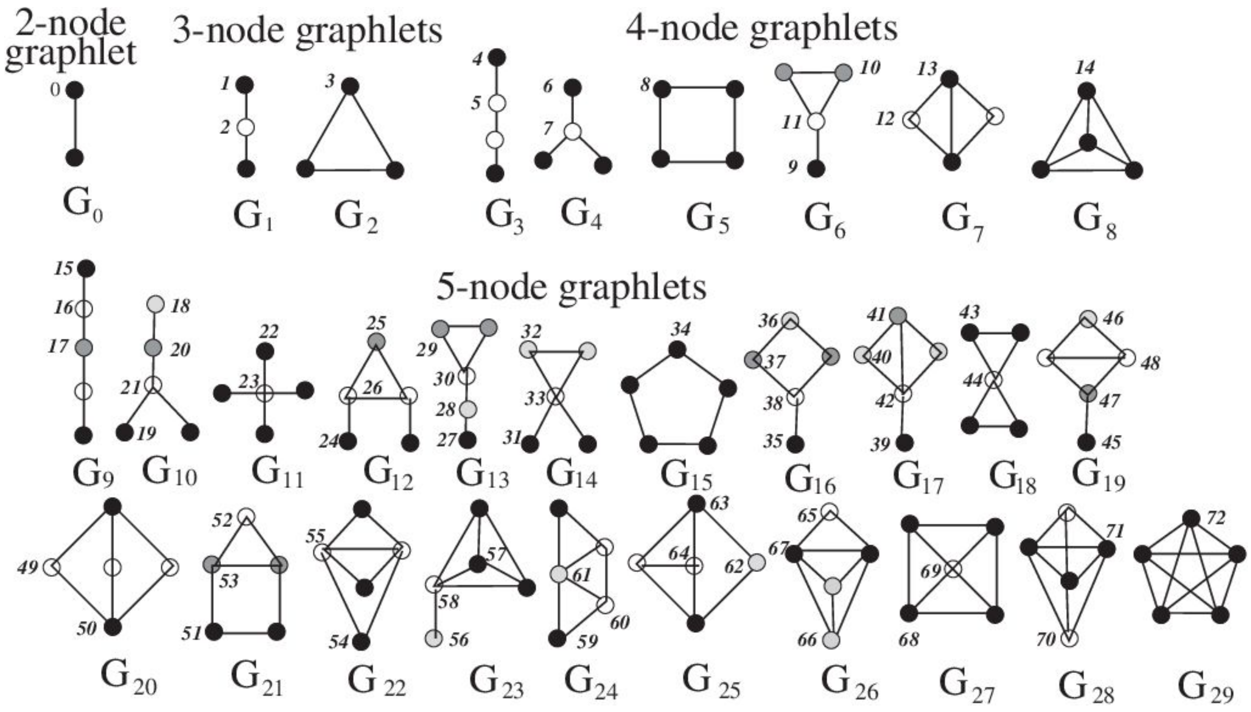


73 different graphlets for node size up to 5

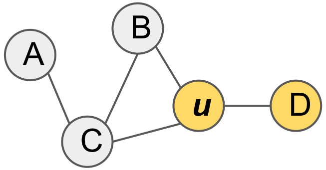
Node features: graphlets

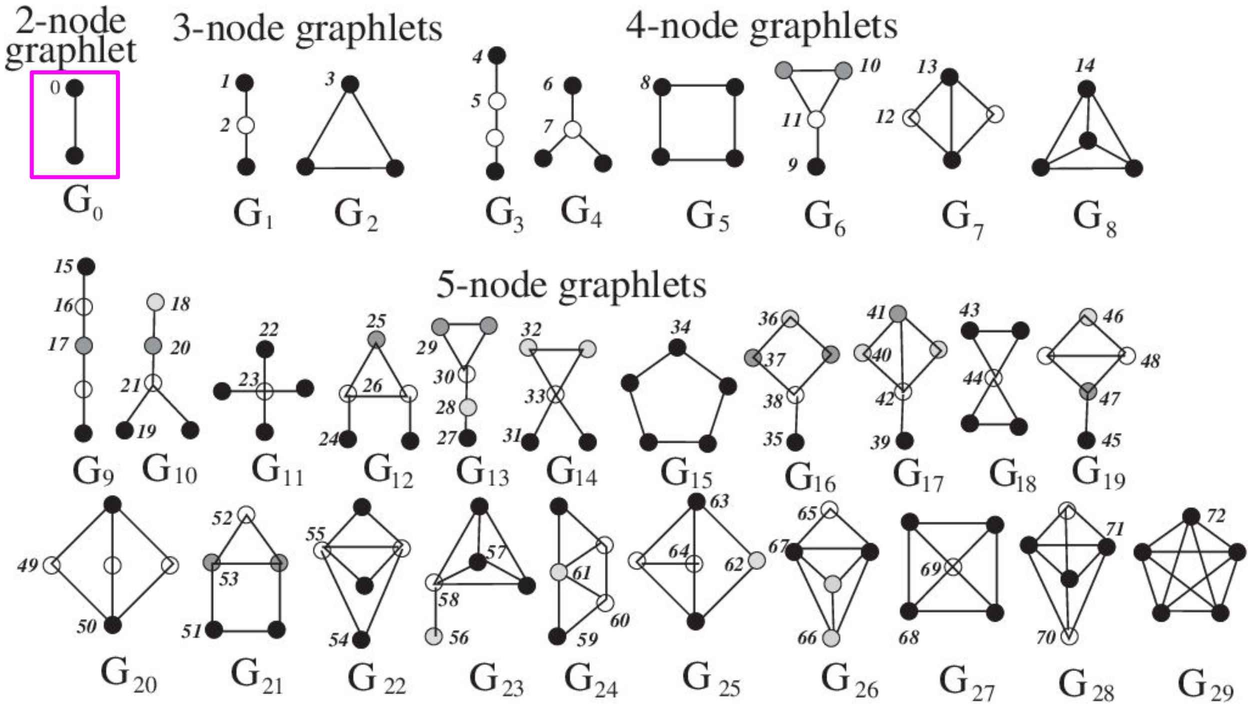


Node u contains graphlets



Node features: graphlets



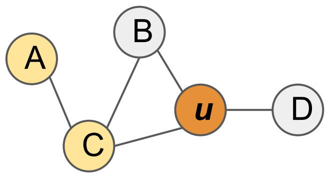


Node ***u*** contains graphlets

- 0,

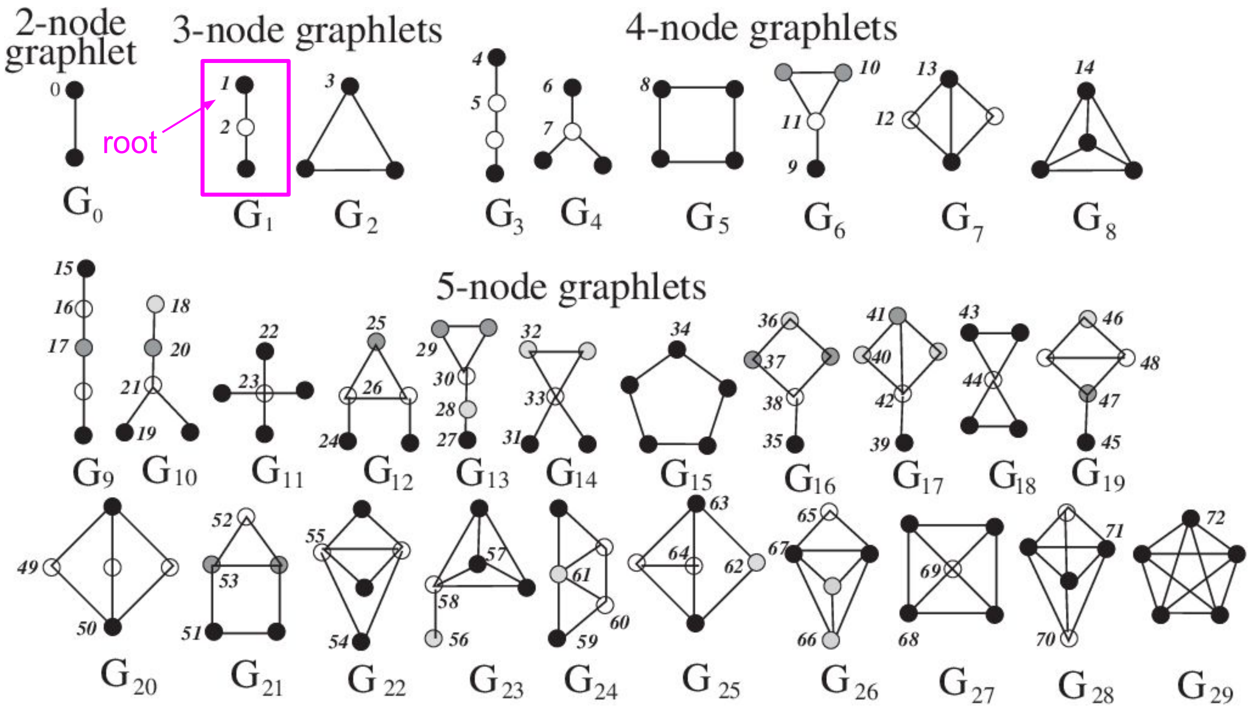
(u-B, u-C are also examples of the 0-th graphlet)

Node features: graphlets

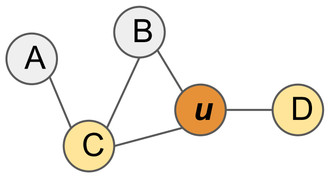


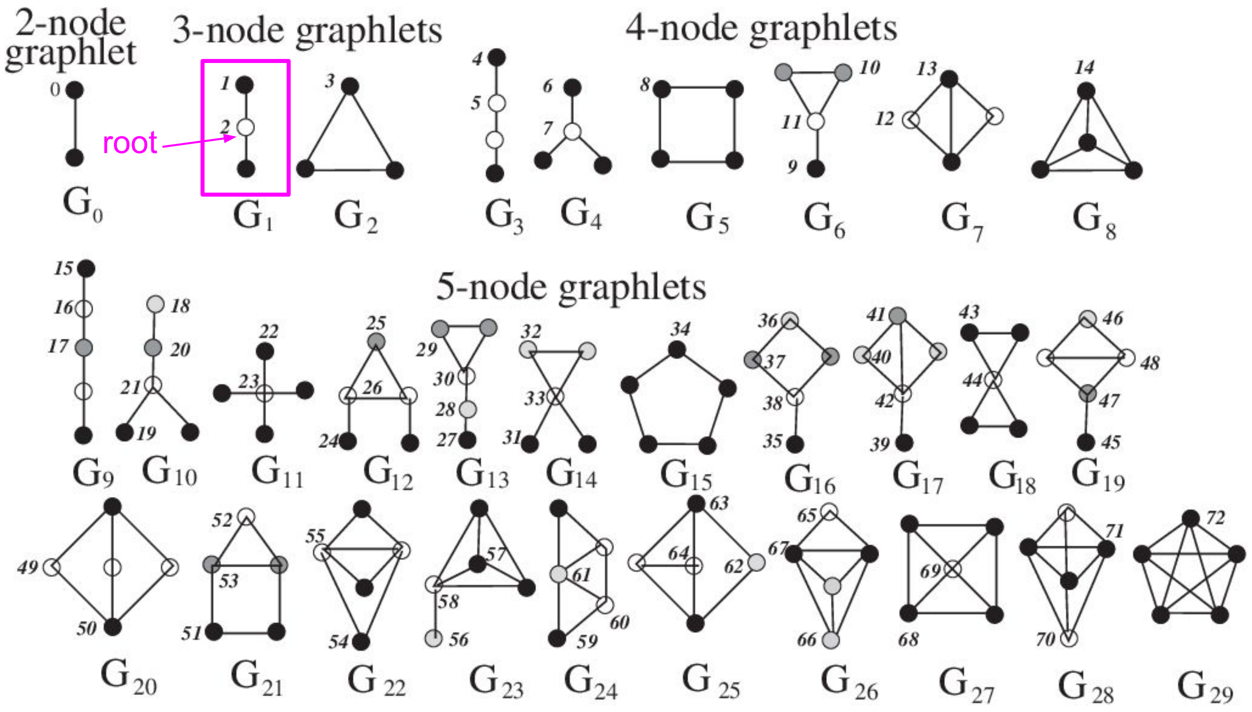
Node ***u*** contains graphlets

- 0, 1



Node features: graphlets

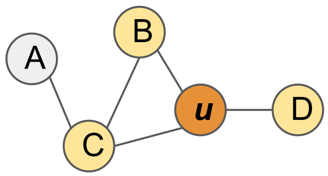




Node ***u*** contains graphlets

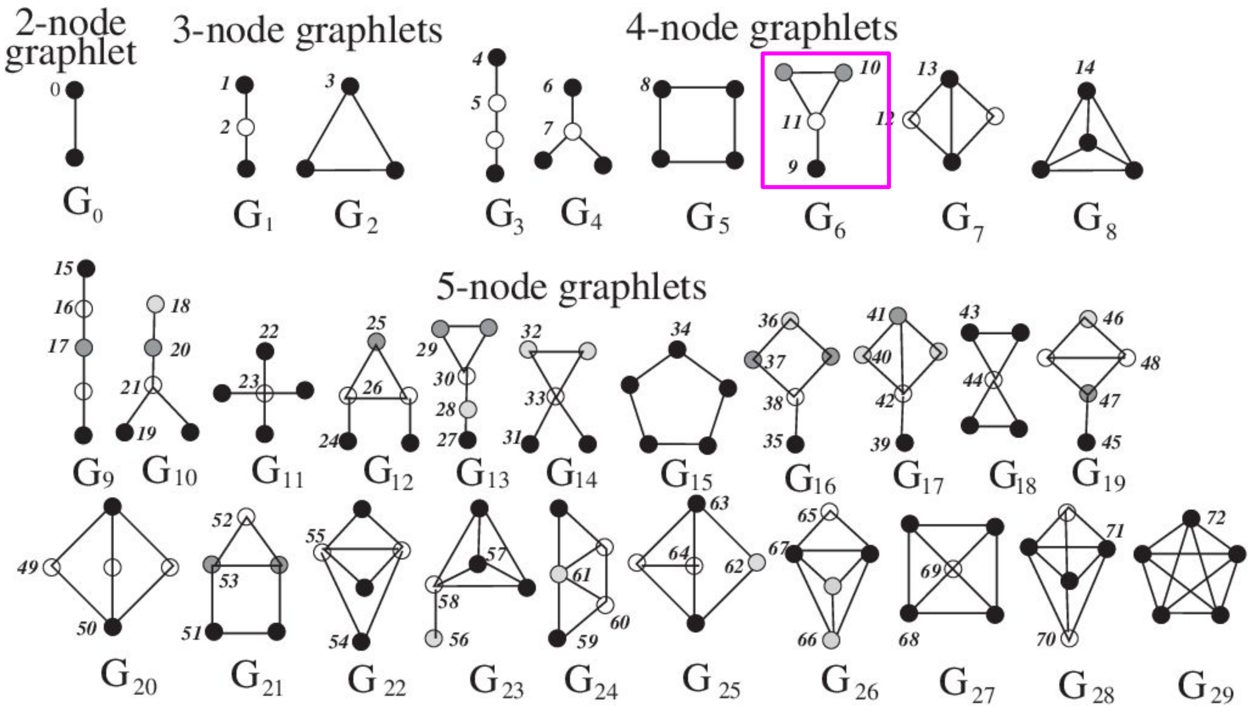
- 0, 1, 2

Node features: graphlets



Node ***u*** contains graphlets

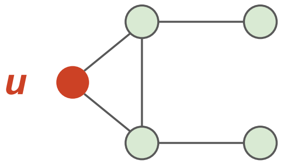
- 0, 1, 2, 3, 5, 10, 11, …

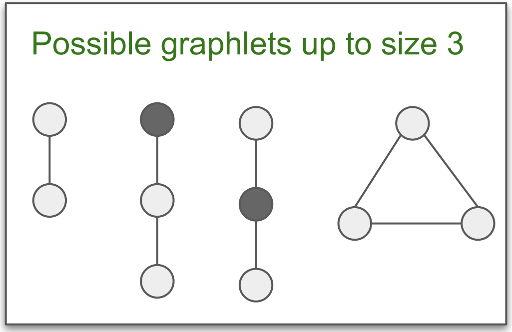


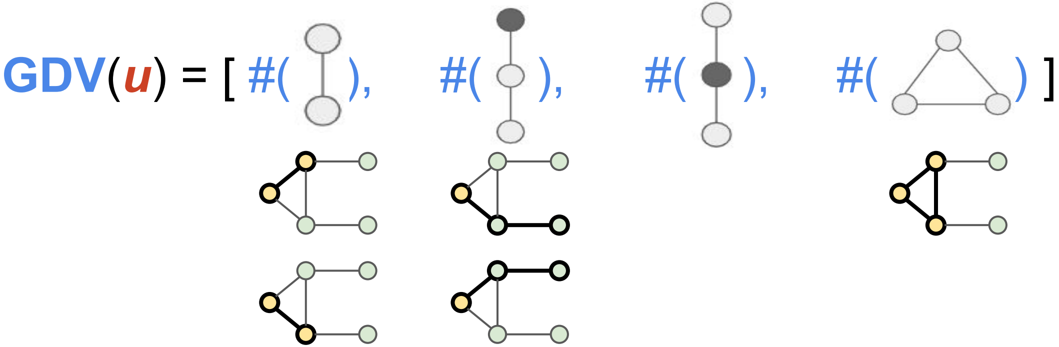
Graphlet Degree Vector (GDV)

- **Graphlet degree vector (GDV)**: a vector that counts the number of graphlet a node touches (for each of the pre-specified graphlet)

- **Example**: Consider GDV of node *u* for graphlets up to size 3







GDV(*u*) = [2, 2, 0, 1]

Graphlet Degree Vector (GDV)

- **Graphlet degree vector (GDV)**: a vector that counts the number of graphlet a node touches (for each of the pre-specified graphlet)
  - Can extend to more nodes (e.g., 73 dimensions if using 2-5 nodes)
  - GDV is a **signature** of a node that describing the topology of its neighborhood

Analogy:

- **Degree** counts #(**edges**) that a node touches
- **Clustering coefficient** counts #(**triangles**) that a node touches
- **GDV** counts #(**graphlets**) that a node touches

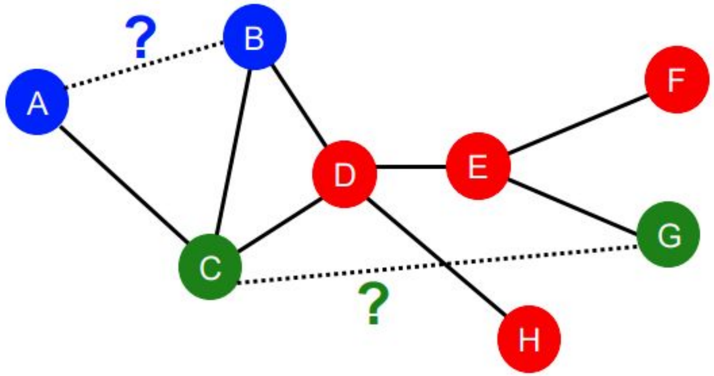
Node-level features: summary

- **Importance-based** features
  - Node degree
  - Node centrality
    - Eigenvector centrality
    - Betweenness centrality
    - Closeness centrality
- **Structure-based** features
  - Node degree
  - Clustering coefficient
  - Graphlet degree vector (GDV)

Edge-level tasks and features

Recap: link-level prediction task

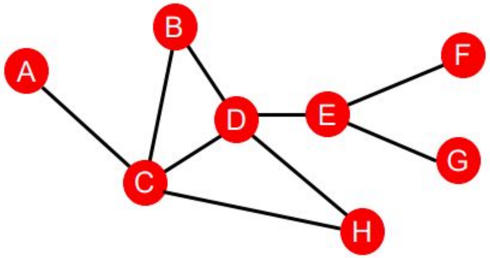
- **Link prediction**: predict new links based on existing links
- The key is to design feature for a pair of nodes



Distance-based features

Shortest-path distance between two nodes

- **Example**:



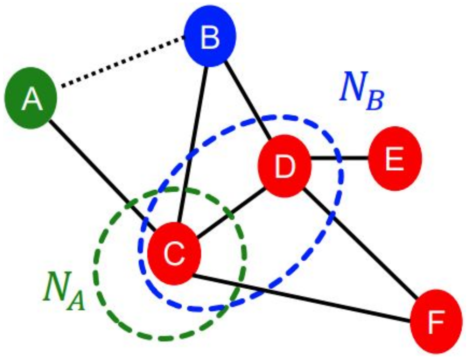
$$S_{BH} = S_{BE} = S_{AB} = 2$$
$$S_{BG} = S_{BF} = 3$$

- **Limitation**: does not capture the degree of neighborhood overlap
  - (B, H) have 2 shared neighbors, while (B, E) and (A, B) only have 1 such node

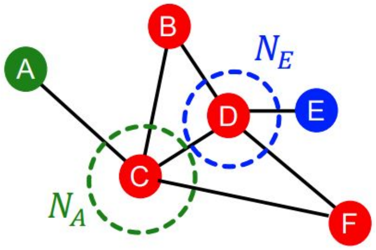
Local neighborhood overlap

#neighboring nodes shared between two nodes

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



- **Limitation:** Metric is always zero if the two nodes do not have any neighbors in common
  - However, the two nodes may still be connected in the future potentially



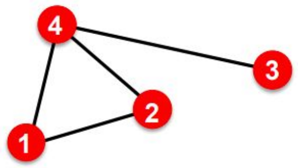
$N_A \cap N_E = \phi$
$|N_A \cap N_E| = 0$

Global neighborhood overlap

Katz index: #(walks) of all lengths between two nodes

- **Q**: how to compute #(walks) between two nodes?
- **A**: Use power of the graph adjacency matrix
  - **Recall:** $\boldsymbol{A}_{uv} = 1$ if $u \in N(v)$
  - Let $\boldsymbol{P}_{\boldsymbol{uv}}^{(\boldsymbol{K})}$ = #walks of length $\boldsymbol{K}$ between $\boldsymbol{u}$ and $\boldsymbol{v}$
  - We will show $\boldsymbol{P}^{(\boldsymbol{K})} = \boldsymbol{A}^{\boldsymbol{k}}$
  - $\boldsymbol{P}_{\boldsymbol{uv}}^{(\boldsymbol{1})}$ = #walks of length 1 (direct neighborhood) between u and v = $\boldsymbol{A}_{\boldsymbol{uv}}$

$$\boldsymbol{P}_{\boldsymbol{12}}^{(\boldsymbol{1})} = \boldsymbol{A}_{\boldsymbol{12}}$$



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

Global neighborhood overlap

Katz index: #(walks) of all lengths between two nodes

- **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

$P_{12}^{(2)} = A_{12}^2$

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

Power of adjacency

Global neighborhood overlap

Katz index: #(walks) of all lengths between two nodes

- **Q**: how to compute #(walks) between two nodes?
- **A**: Use power of the graph adjacency matrix
  - $\boldsymbol{A_{uv}}$ specifies #(walks) of length 1 (direct neighbor) between $\boldsymbol{u}$ and $\boldsymbol{v}$
  - $\boldsymbol{A_{uv}^{(2)}}$ specifies #(walks) of length 2 (neighbor of neighbor) between $\boldsymbol{u}$ and $\boldsymbol{v}$
  - …
  - $\boldsymbol{A_{uv}^{(L)}}$ specifies #(walks) of length L between $\boldsymbol{u}$ and $\boldsymbol{v}$

Global neighborhood overlap

Katz index: #(walks) of all lengths between two nodes

Sum over all walk lengths

$$S_{u,v} = \sum_{l=1}^{\infty} \boxed{\beta^l} \boxed{A_{u,v}^l}$$

#walks of length ***l*** between ***u*** and ***v***

0<β<1: discount factor

- Katz index can be computed in closed-form:

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

by geometric series of matrices

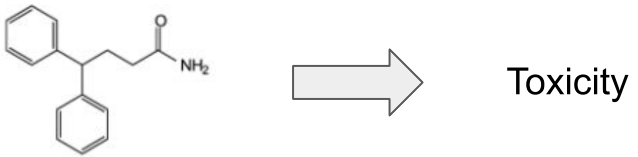
Summary of link-level features

- **Distance-based features**
  - Shortest-path distance between two nodex
    - Does not capture neighborhood overlap
- **Local neighborhood overlap**
  - #nodes shared between two nodes
    - Becomes 0 when no shared neighbors
- **Global neighborhood overlap**
  - Katz index: #walks of all lengths between two nodes
    - Captures global graph structure

Graph-level tasks and features

Graph-level tasks and features

- **Graph-level task**: predict property of the entire graph



- **Goal of graph-level feature**
  - We want features that characterize the structure of an entire graph
- Example in this lecture: **graph kernels**
  - Measure similarity between two graphs

Background: Kernel methods

- **Kernel methods** are widely-used in machine learning
- **Idea**: Design kernels (similarity functions) 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.

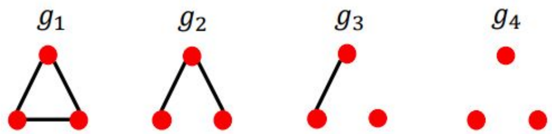
Graph kernels

- **Kernel kernels** measure similarity between two graphs
- Two examples in this lecture
  - Graphlet kernel [1]
  - Weisfeiler-Lehman kernel [2]
- Other graph kernels
  - Random-walk kernel
  - Shortest-path graph kernel
  - …

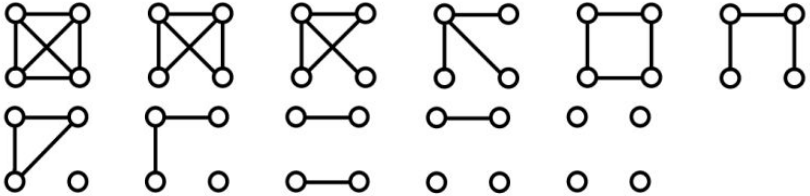
[1] Shervashidze, Nino, et al. "Efficient graphlet kernels for large graph comparison." Artificial Intelligence and Statistics. 2009.
[2] Shervashidze, Nino, et al. "Weisfeiler-lehman graph kernels." Journal of Machine Learning Research 12.9 (2011).

Graphlet kernel

- **Idea**: count the #(different graphlets) in a graph
- Let $G_k = (g_1, g_2, \ldots, g_{n_k})$ 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

Note: definition of graphlets here is slightly different from node-level features

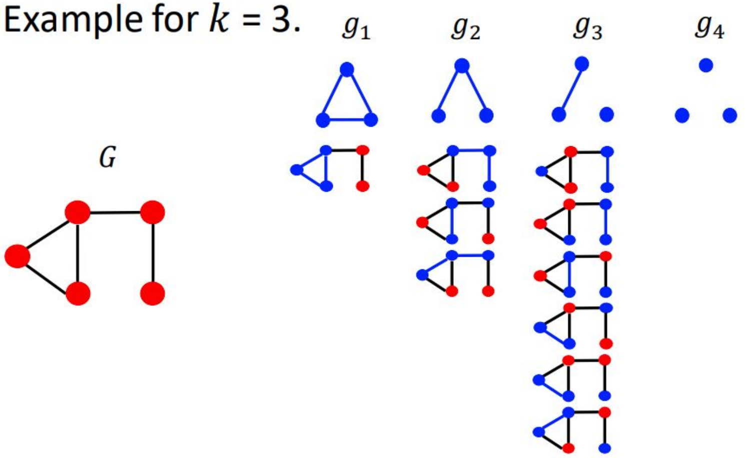
- Nodes in graphlets do not need to be connected
- Graphlets here are not rooted

Graphlet features

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

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

- Example for k = 3.



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

Graphlet kernel

- Given two graphs, $\boldsymbol{G}$ and $\boldsymbol{G'}$, graphlet kernel is computed as

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

- **Normalization**: if $\boldsymbol{G}$ and $\boldsymbol{G'}$ have different sizes, that will greatly skew the value, so normalize each feature vector:

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

- **Limitation**: counting graphlets is computationally expensive
  - Counting size-k graphlets for a graph with size n by enumeration takes n^k

Weisfeiler-Lehman Kernel

- **Idea**: use neighborhood structure to iteratively enrich node vocabulary
  - The 1-dim Weisfeiler-Lehman (WL) algorithm is commonly known as **color refinement**
- **Algorithm:**
  - **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

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

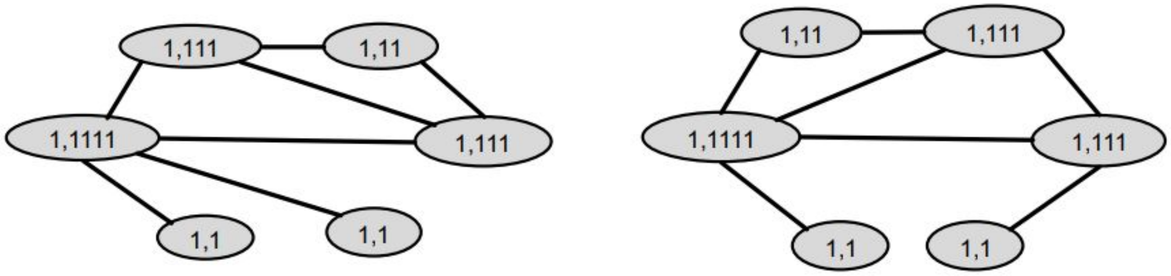
      where HASH maps different inputs to different colors.
    - After K steps of color refinement, $c^{(K)}(v)$ summarizes the structure of K-hop neighborhood

Color refinement

Example of color refinement given two graphs

- Assign initial colors

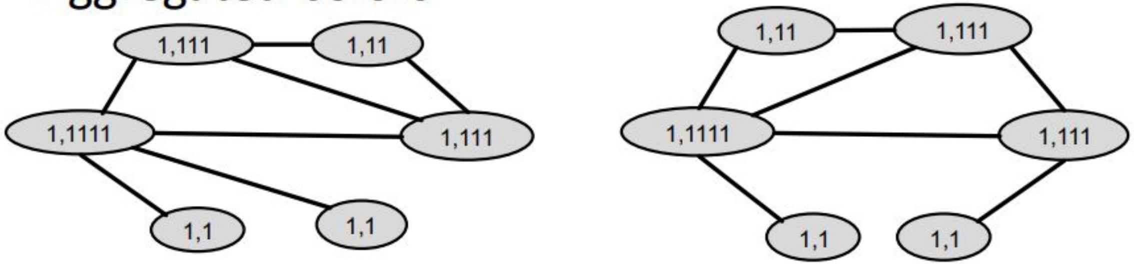
- Aggregate neighboring colors



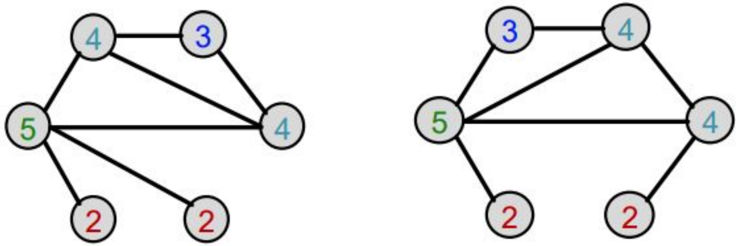
Color refinement

Example of color refinement given two graphs

- Aggregated colors



- Hash aggregated colors



Hash table

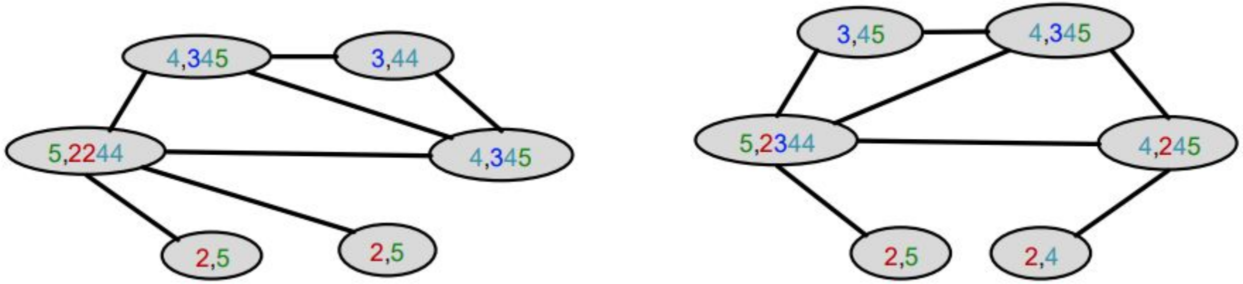
1,1	-->	2
1,11	-->	3
1,111	-->	4
1,1111	-->	5

Color refinement

Example of color refinement given two graphs

- Aggregated colors

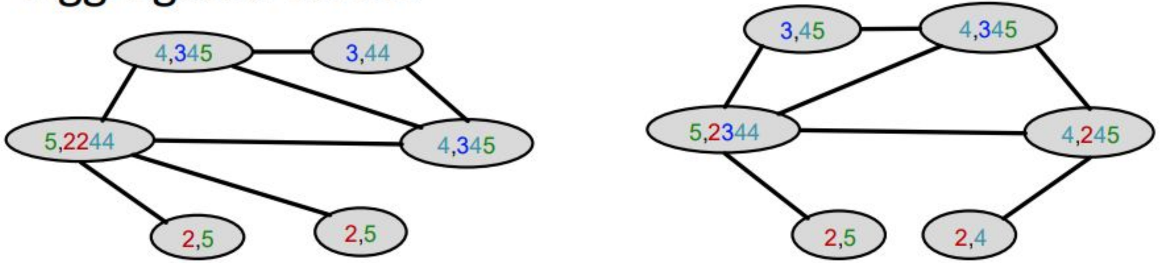
- Hash aggregated colors



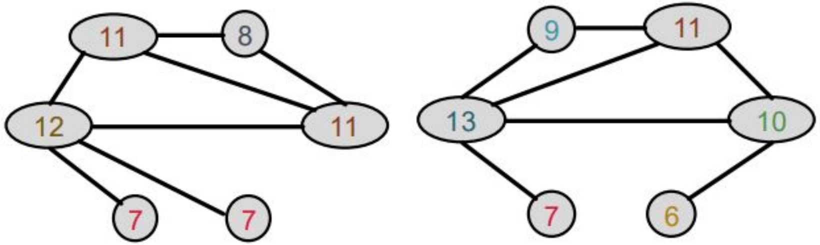
Color refinement

Example of color refinement given two graphs

- Aggregated colors



- Hash aggregated colors

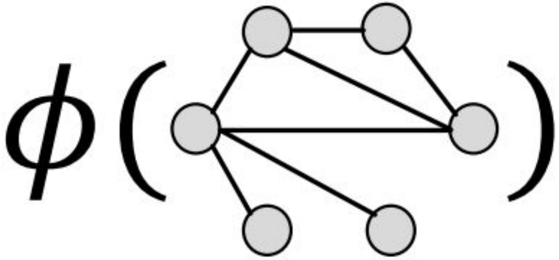


Hash table

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

Weisfeiler-Lehman graph features

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

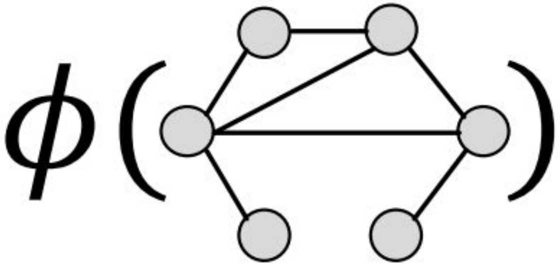


Colors

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

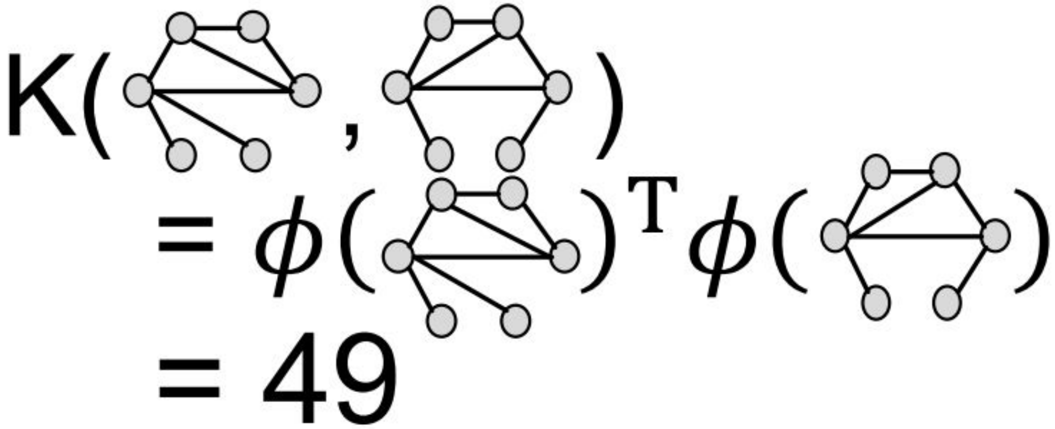


1,2,3,4,5,6,7,8,9,10,11,12,13

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

Weisfeiler-Lehman kernel

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



Weisfeiler-Lehman kernel

- 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 of graph-level features

- **Graphlet kernel**
  - Count number of different graphlets
    - Computationally expensive
- **Weisfeiler-Lehman kernel**
  - K-step color refinement algorithm
    - Computationally efficient

Summary of today

- Network basics
- Traditional ML pipelines for graphs
    - Hand-crafted feature + ML model
- Node-level features
- Link-level features
- Graph-level features