

# Node-level and Edge-level Tasks

Prof. O-Joun Lee

Dept. of Artificial Intelligence,  
The Catholic University of Korea  
*ojlee@catholic.ac.kr*



네트워크 과학 연구실  
NETWORK SCIENCE LAB



가톨릭대학교  
THE CATHOLIC UNIVERSITY OF KOREA



# Contents

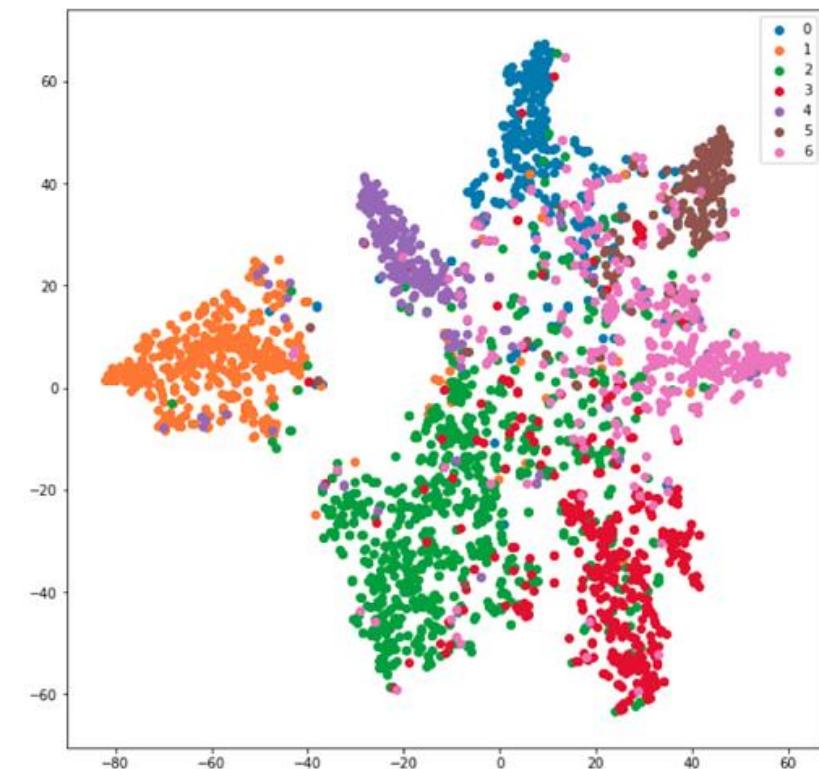
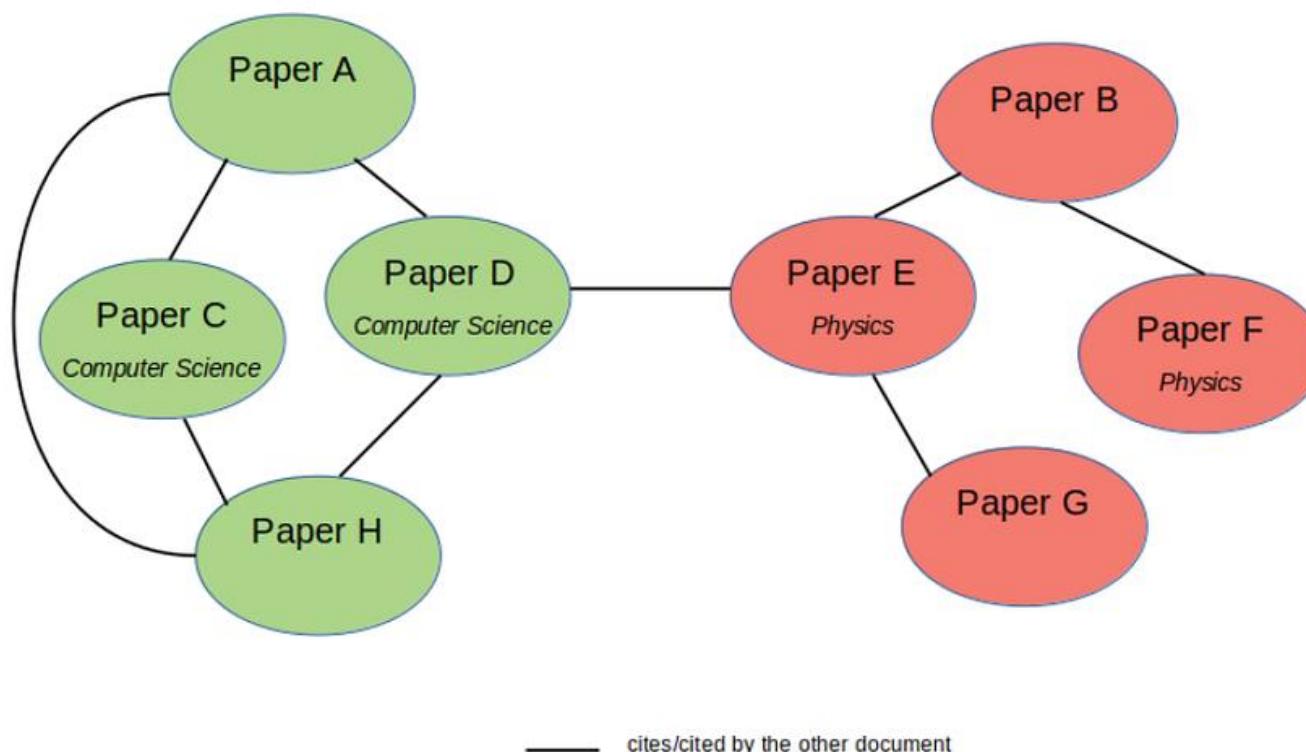


- Overview of Machine Learning (ML) pipeline
- Feature Extraction Methods: How to extract structures for each node?
  - Node-level Features
    - Local features: Node Degree, Centrality, Clustering Coefficient, Neighboring Information
    - Structure-based features: Graphlet, Similarity, Label Propagation, Embeddings
  - Edge-level Features:
    - Distances, Local and Global features
- Node Classification Task
- Evaluation Metrics

## ➤ One of main ML task: Classification

### ➤ Node-level Classification:

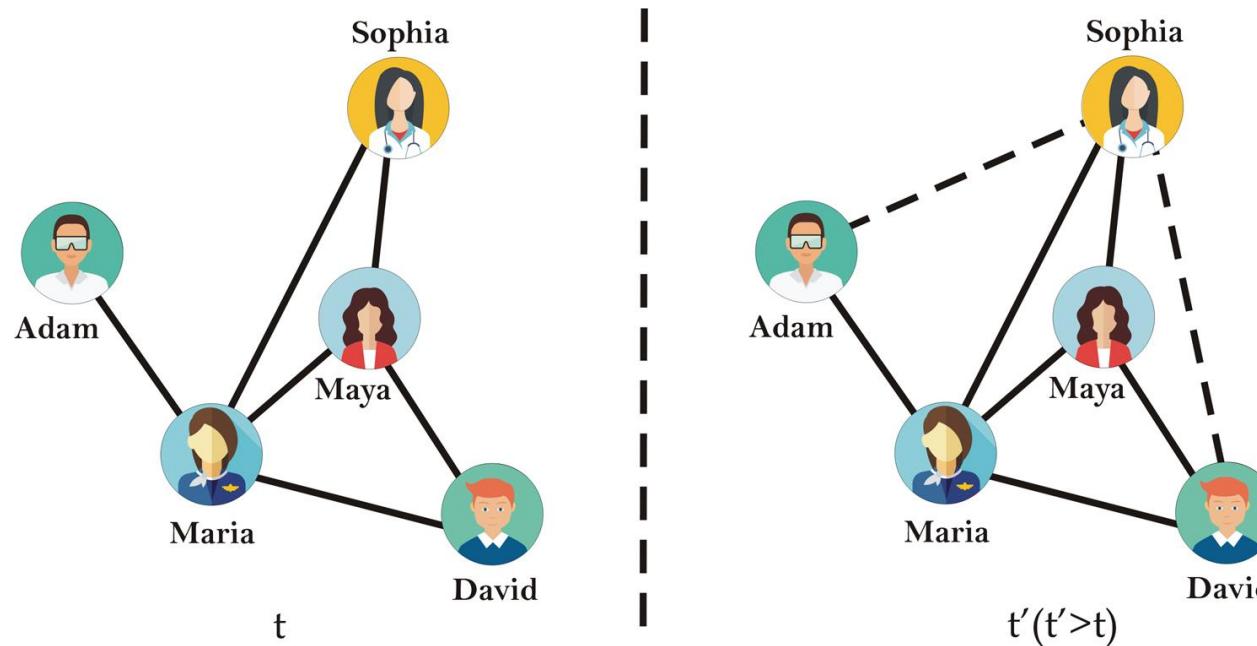
- Predict the classes or labels of nodes
- For example, detect the paper field in a citation network



➤ One of main ML task: Classification

➤ Edge-level or Link-level Classification:

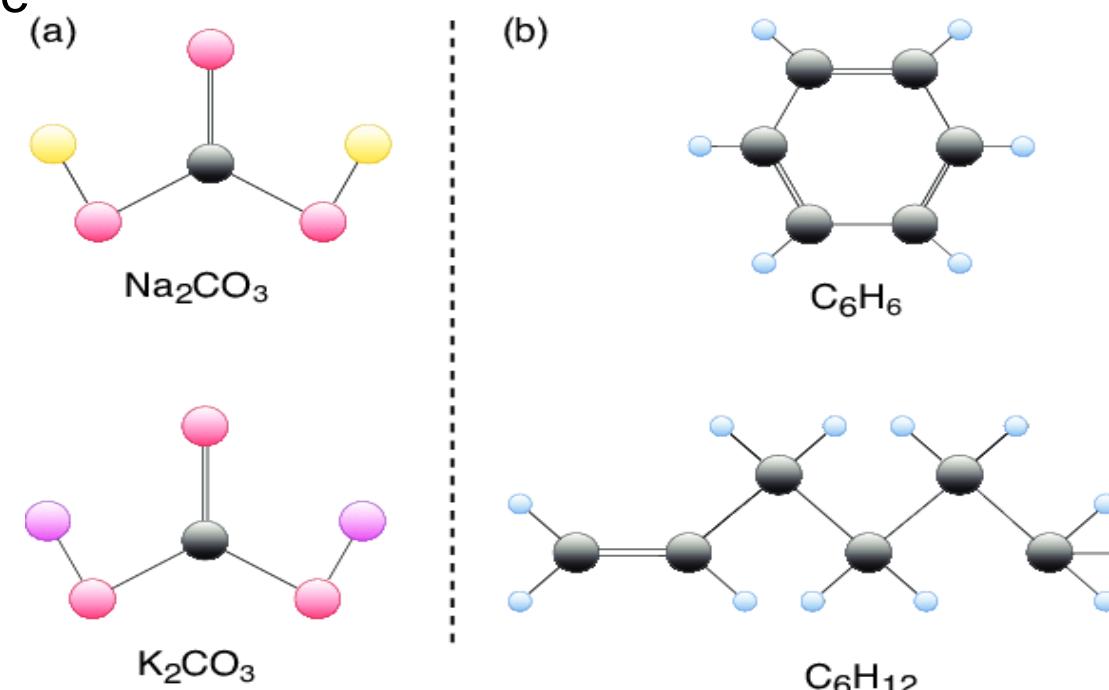
- Predict the classes or labels of nodes
- For example, a social networking service suggests possible friend connections based on network data



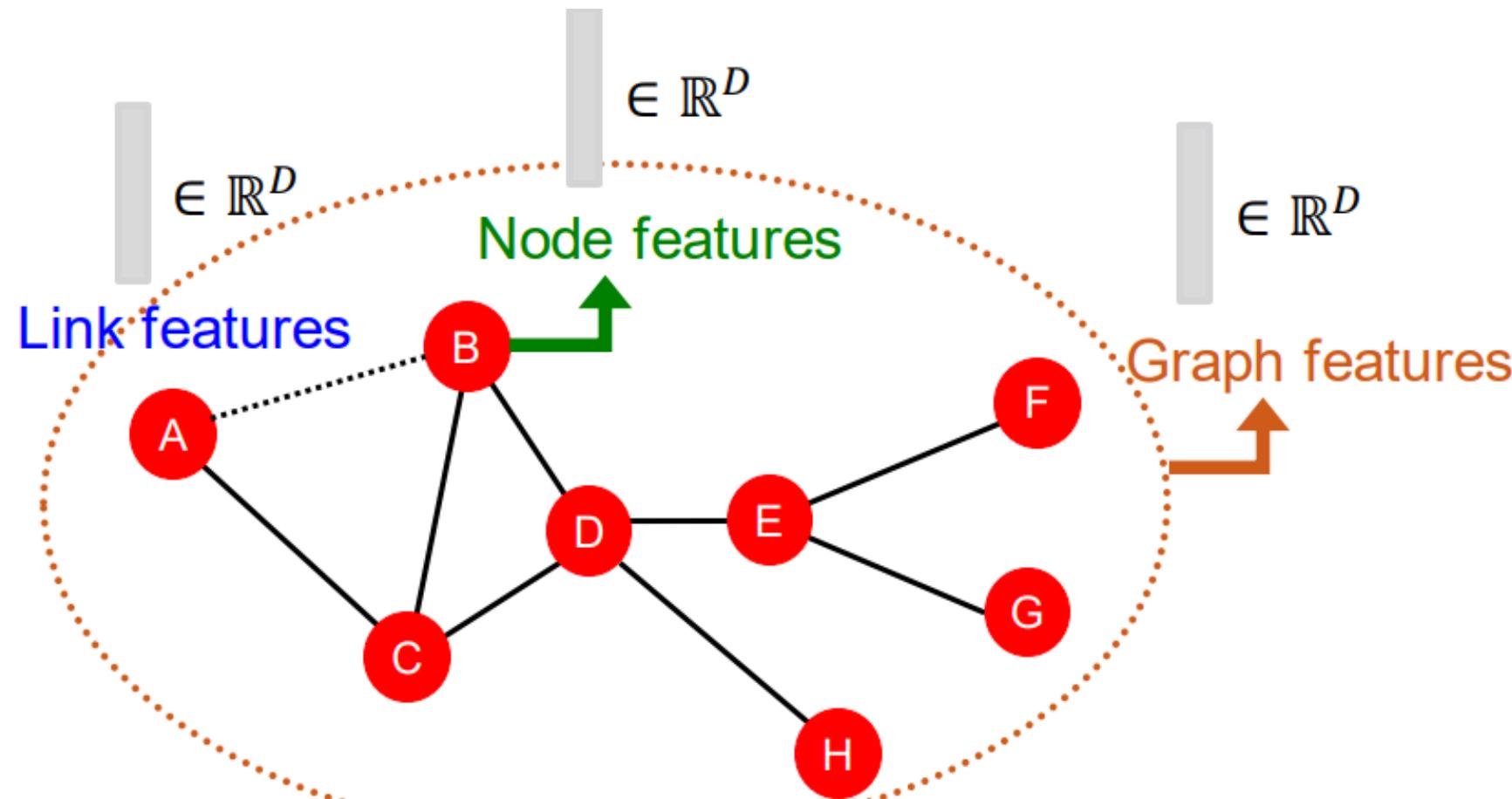
## ➤ One of main ML task: Classification

### ➤ Graph-level Classification:

- Classify a graph itself into different categories
- Input: A collection of graphs
- For example, determine if a chemical compound is toxic or non-toxic by looking at its graph structure



- Design features for nodes/edges(links)/graphs
- Obtain features for all training data



➤ **Feature extraction:**

- Node features
- Edge features
- Graph features

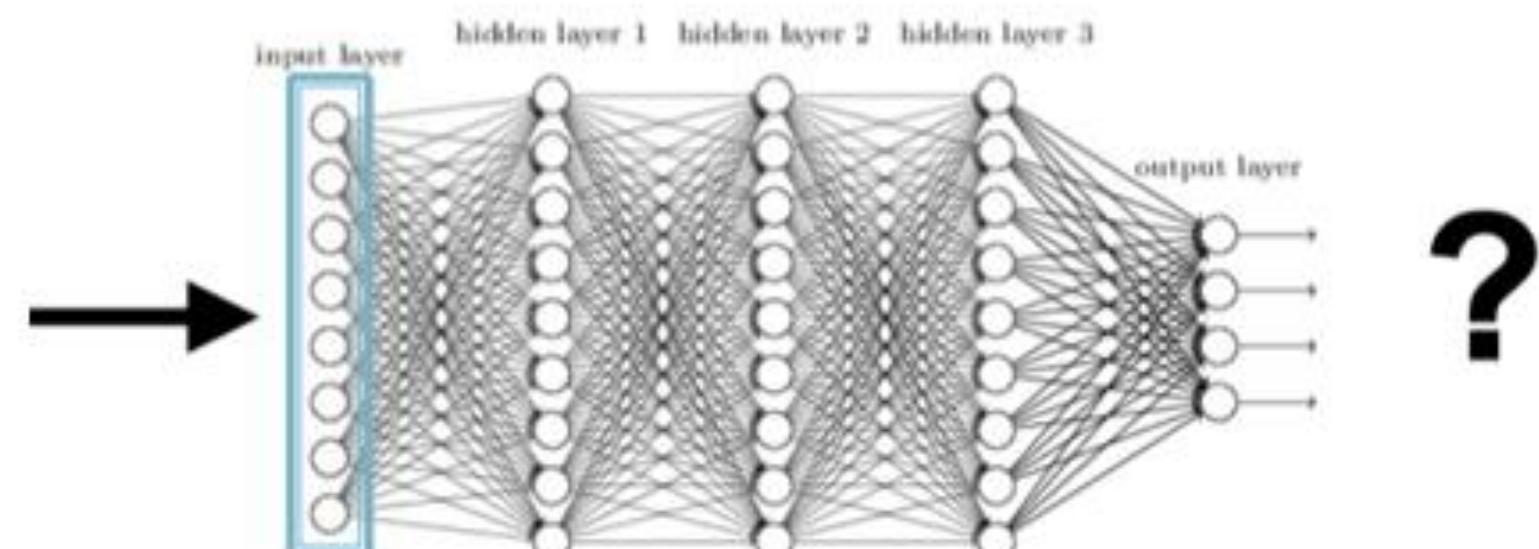
$$\begin{array}{l} \text{Node features} \\ \in \mathbb{R}^D \end{array}$$
$$\begin{array}{l} \text{Edge features} \\ \in \mathbb{R}^D \end{array}$$
$$\begin{array}{l} \text{Graph features} \\ \in \mathbb{R}^D \end{array}$$

➤ **Train a ML model:**

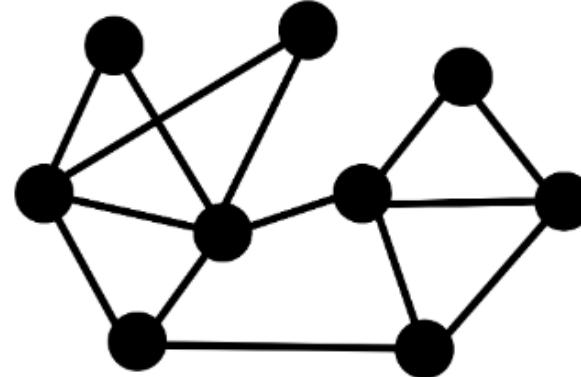
- Logistic Regression
- Random forest
- Neural network, etc.

➤ **Apply the model:**

- Given a new node/link/graph, obtain its features and make a prediction



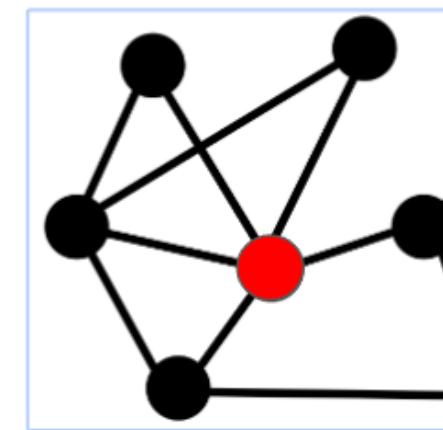
- Using effective features over graphs is the key to achieve good model performance
- Beside the original node/edge/graph features, can we embed topology structure into node/edge/graph features?



Graph



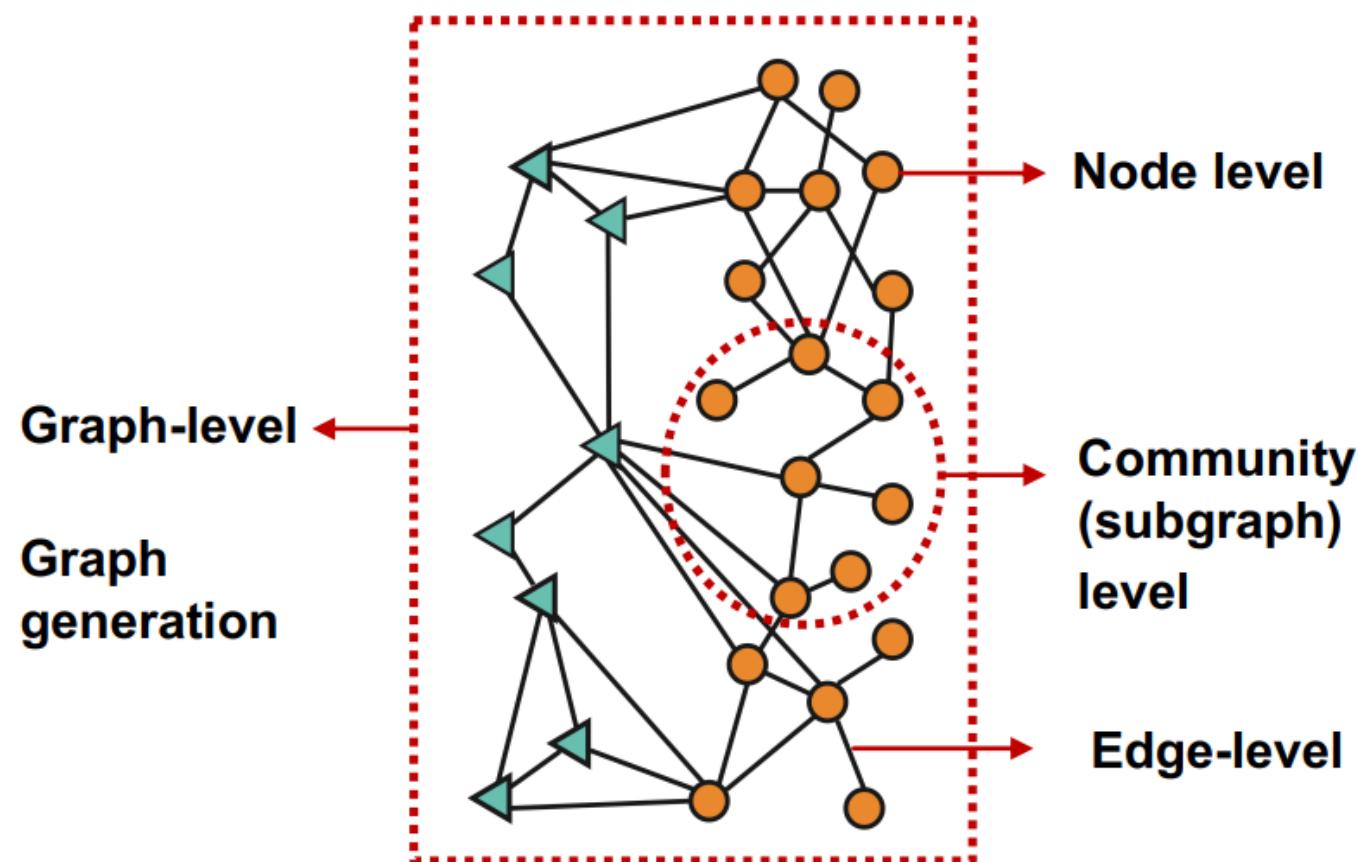
Node prediction



With neighboring topology

➤ Three main types of graph components:

1. Node-level features
2. Edge-level features
3. Graph-level features (2025/05/20)c



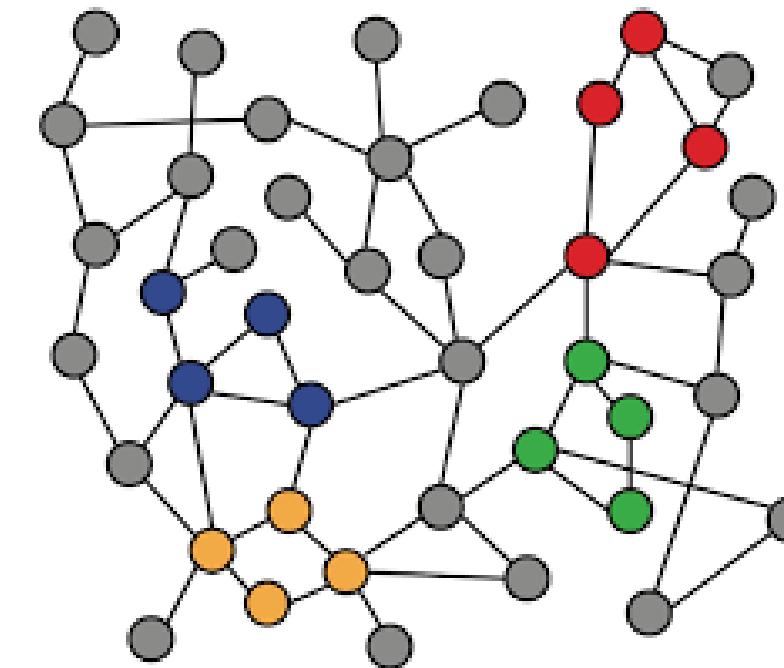
➤ **Goal:** Characterize the structure and position of a node in the network:

➤ Local features:

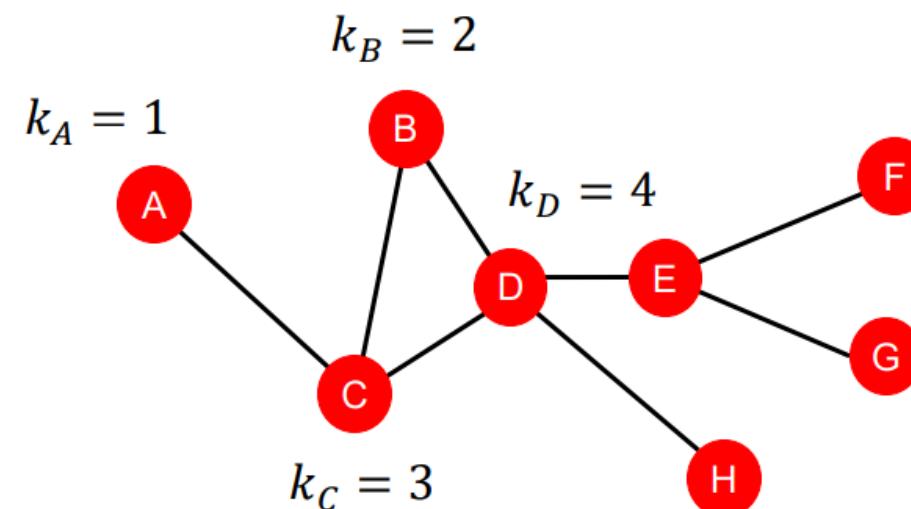
- Node degree
- Node centrality
- Vector similarity
- Clustering coefficient
- Neighbouring information

➤ Structure-based features:

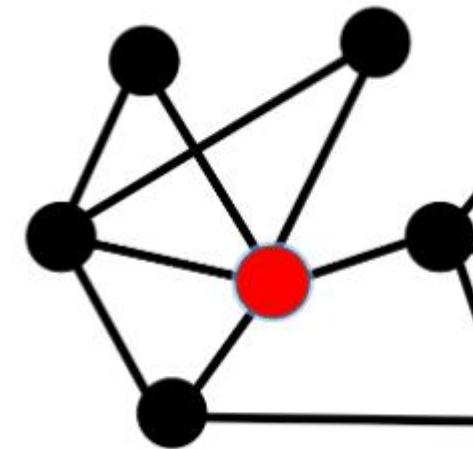
- Graphlets
- SimRank
- Label Propagation



- The degree  $k_v$  of node  $v$  is the number of edges (neighboring nodes) the node has.
  - In-degree: Number of incoming edges to the node
  - Out-degree: Number of outgoing edges from the node
  - Total degree: Sum of in-degree and out-degree
- Treats all neighboring nodes equally



- Node degree counts the neighbouring nodes without capturing their importance
- Node centrality  $c_v$ , takes the node importance in a graph into account
- Different ways to evaluate importance:
  - Eigenvector centrality
  - Betweenness centrality
  - Closeness centrality
  - and many others....



## ➤ Eigenvector Centrality

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

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

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

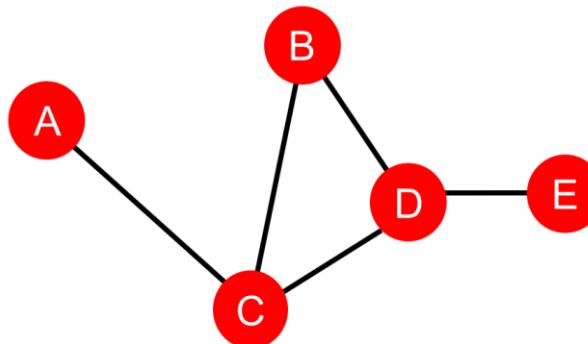
- We can see that centrality  $c$  is the eigenvector for  $A$

## ➤ Betweenness Centrality

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

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

- For example:



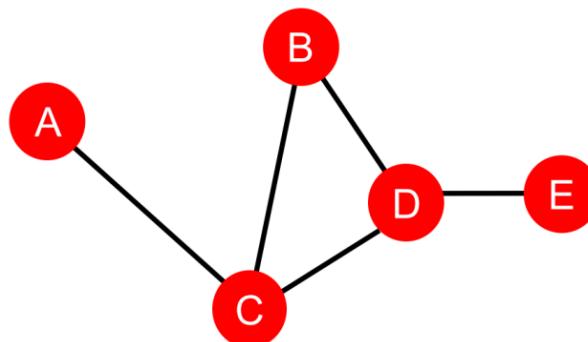
- $C_A = C_B = C_E = 0$
- $C_C = 3$  (A-C-B, A-C-D, A-C-D-E)
- $C_D = 3$  (A-C-D-E, B-D-E, C-D-E)

## ➤ Closeness Centrality

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

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

- For 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)

# Node Features: Node Centrality (4)

## ➤ Katz centrality:

- Compute the centrality for a node based on the centrality of its neighbours
- It is a generalization of the eigenvector centrality
- The Katz centrality for node  $v_i$  is:

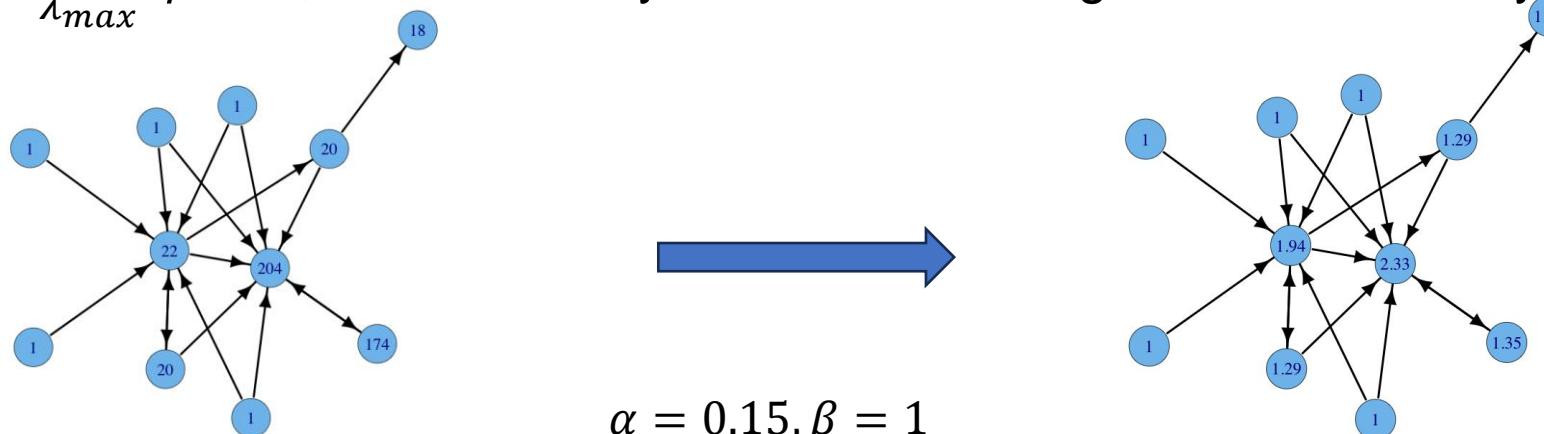
$$x_i = \alpha \sum_j A_{ij} x_j + \beta,$$

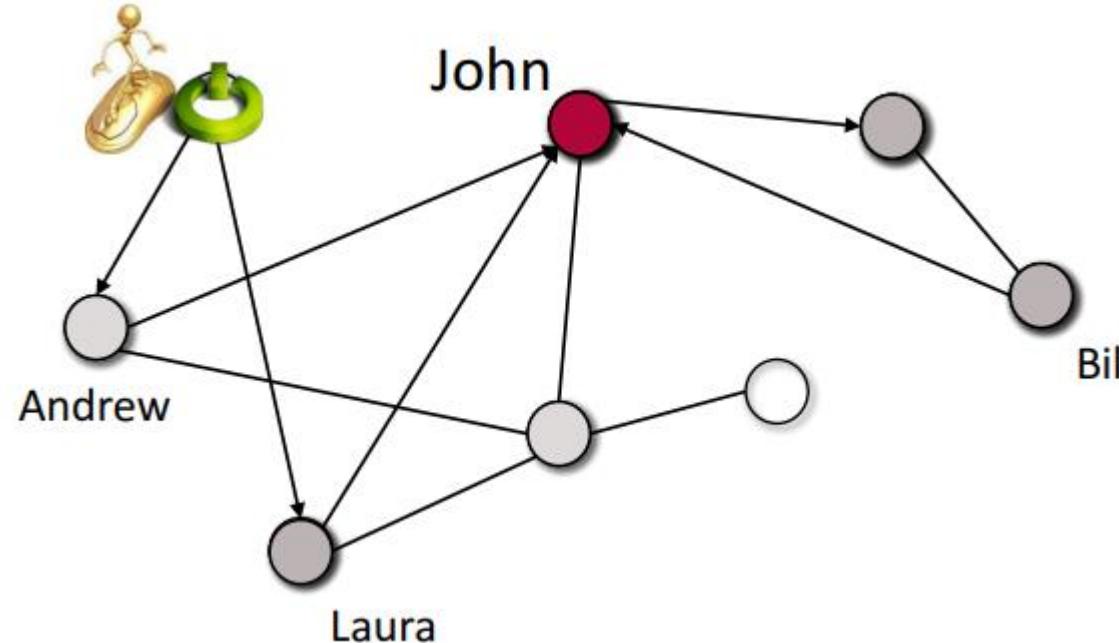
where:

$\alpha$  is a constant called damping factor, and  $\beta$  is a bias constant,

$A$  is the adjacency matrix.

- When  $\alpha = \frac{1}{\lambda_{max}}$ ,  $\beta = 0$ , Katz centrality is the same as eigenvector centrality





- The importance of John is high if Laura, Andrew, and Bill are also important

$$P_{Rank}(John) = \frac{P_{Rank}(Bill) + P_{Rank}(Laura) + P_{Rank}(Andrew)}{\# \text{ of } in-links}$$

- Two nodes are structurally equivalent if they connect to the same set of actors

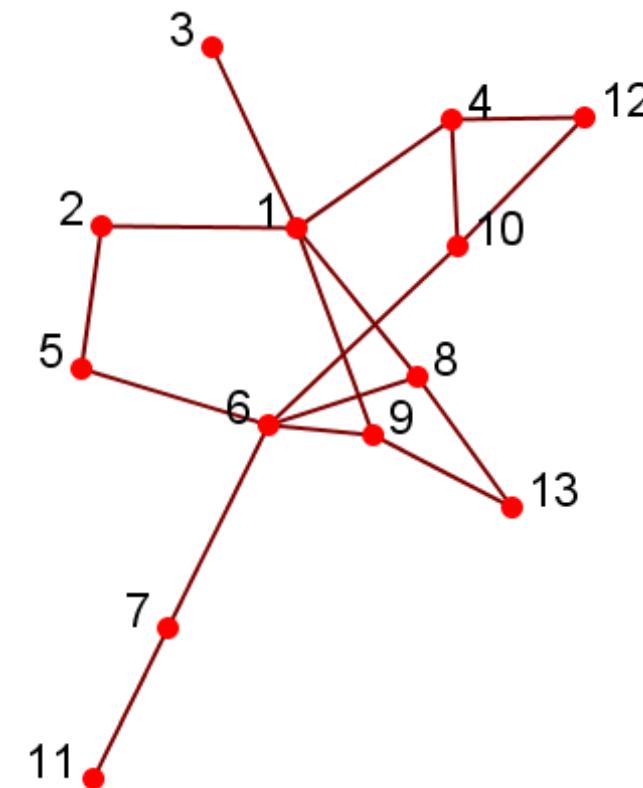
	1	2	3	4	5	6	7	8	9	10	11	12	13
A vector	5	1											
Structurally equivalent	8	1											1
	9	1											1

Cosine Similarity:  $similarity = \cos(\theta) = \frac{A \cdot B}{\|A\| \|B\|}$

$$sim(5,8) = \frac{1}{\sqrt{2} \times \sqrt{3}} = \frac{1}{\sqrt{6}}$$

Jaccard Similarity:  $J(A, B) = \frac{|A \cap B|}{|A \cup B|}$

$$J(5,8) = \frac{|\{6\}|}{|\{1,2,6,13\}|} = 1/4$$



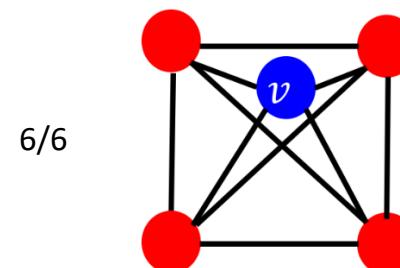
- Measures **how connected v's neighbouring nodes** are:

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

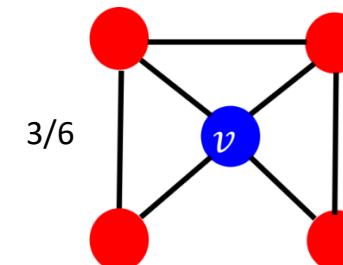
#(node pairs among  $k_v$  neighboring nodes)

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

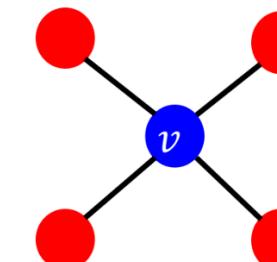
- For example:



$$e_v = 1$$



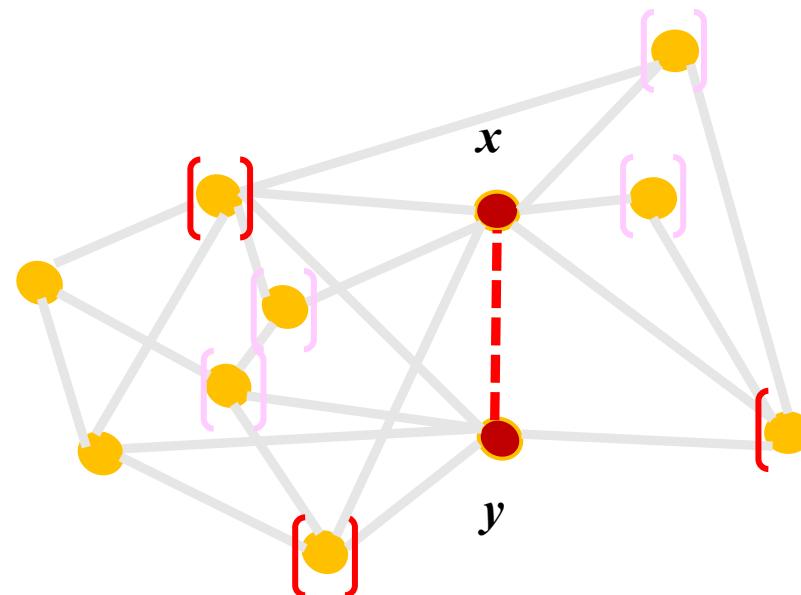
$$e_v = 0.5$$



$$e_v = 0$$

## ➤ Common neighbours:

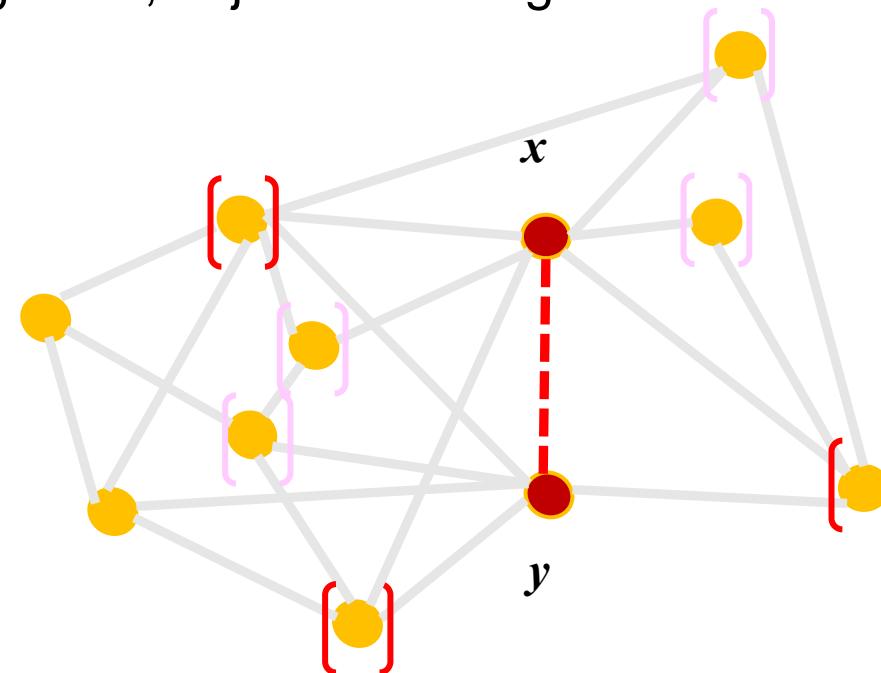
- How many neighbours are in common between  $x$  and  $y$
- $X$  and  $Y$  have 3 common neighbours, more likely to collaborate
- Let  $N(x)$  denote the set of nodes adjacent to  $x$ ,  $N(x) = \{m | (x, m) \in E\}$



$$CN = |N(x)| \cap |N(y)| = 3$$

➤ Jaccard coefficient:

- How likely a neighbour of  $x$  is also a neighbour of  $y$
- Same as common neighbors, adjusted for degree

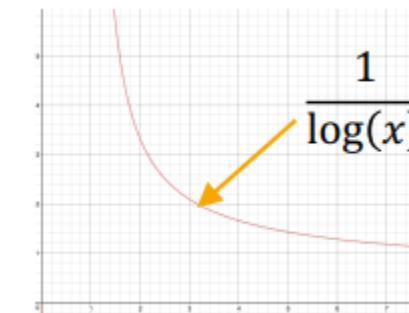
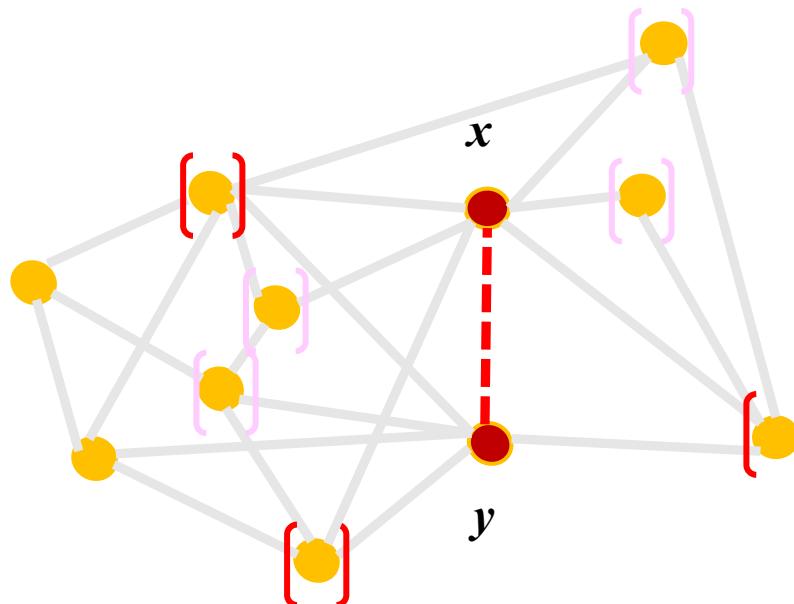


$$JC = \frac{|N(x) \cap N(y)|}{|N(x) \cup N(y)|} = \frac{CN}{d_x + d_y - CN}$$

- Adamic-Adar index:

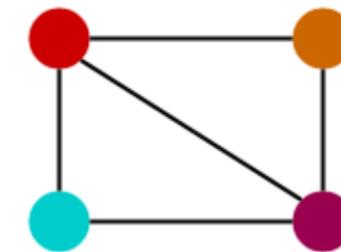
- Large weight to common neighbors with low degree (the lower the degree the higher the relevance)
- E.g., Neighbors who are linked with 2 nodes are assigned weight =  $1/\log(2) = 1.4$
- Neighbors who are linked with 5 nodes are assigned weight =  $1/\log(5) = 0.62$

$$AA = \sum_{z \in CN} \frac{1}{\log d_z}$$

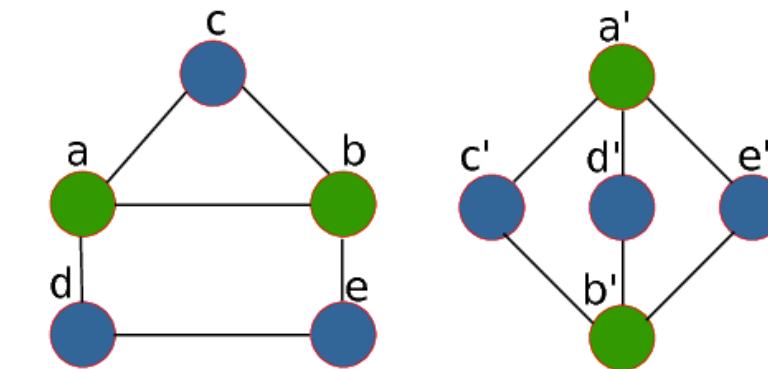


- **Isomorphic graphs** have the same number of nodes, edges, and the same edge connectivity

Graph G	Graph H	An isomorphism between G and H
		$f(a) = 1$ $f(b) = 6$ $f(c) = 8$ $f(d) = 3$ $f(g) = 5$ $f(h) = 2$ $f(i) = 4$ $f(j) = 7$

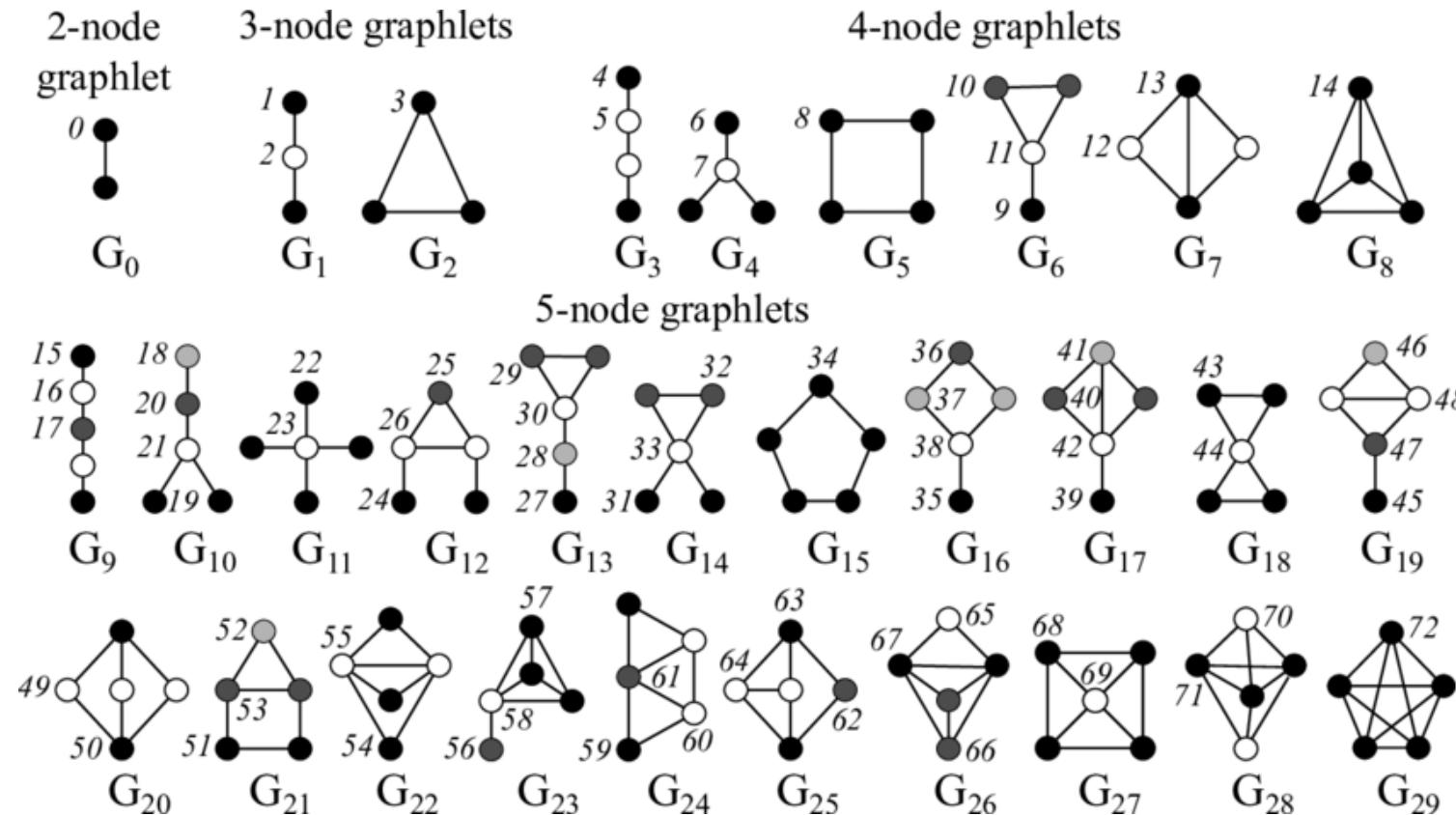


Isomorphic

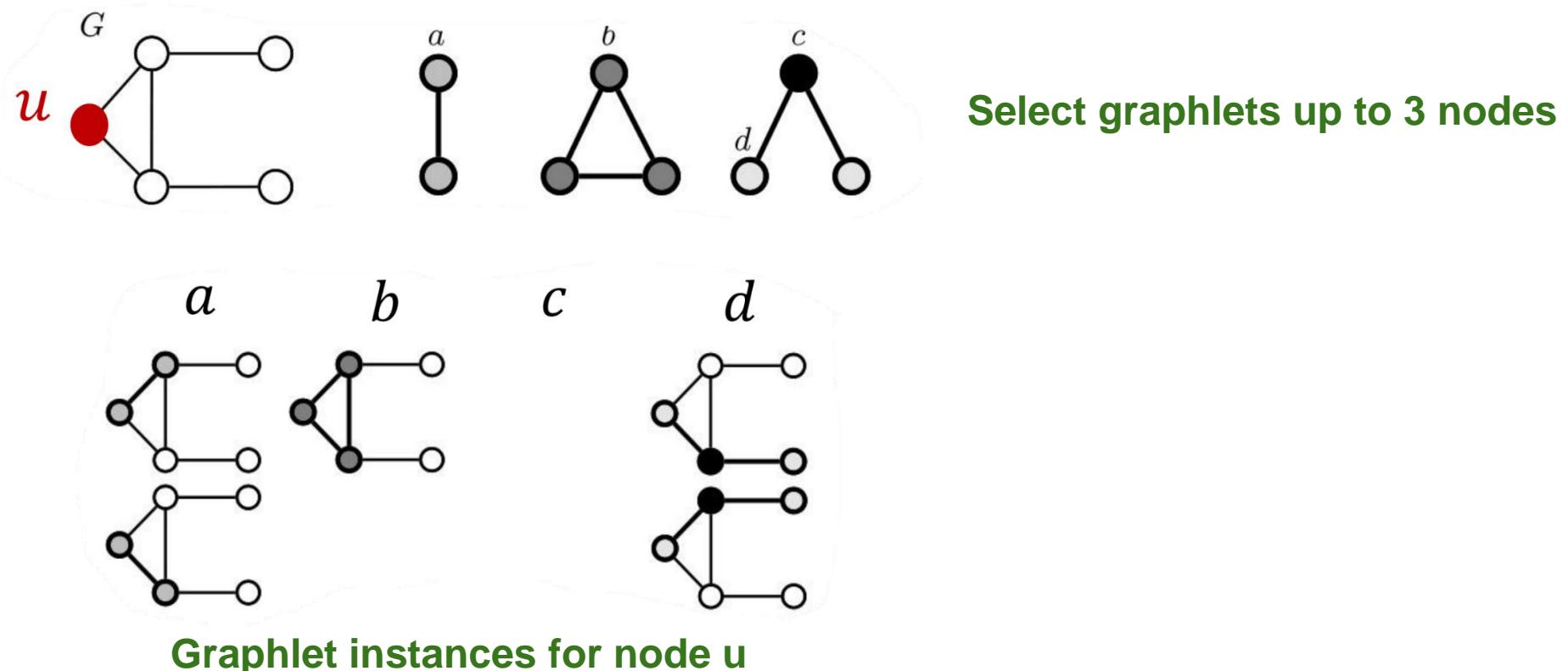


Non-Isomorphic

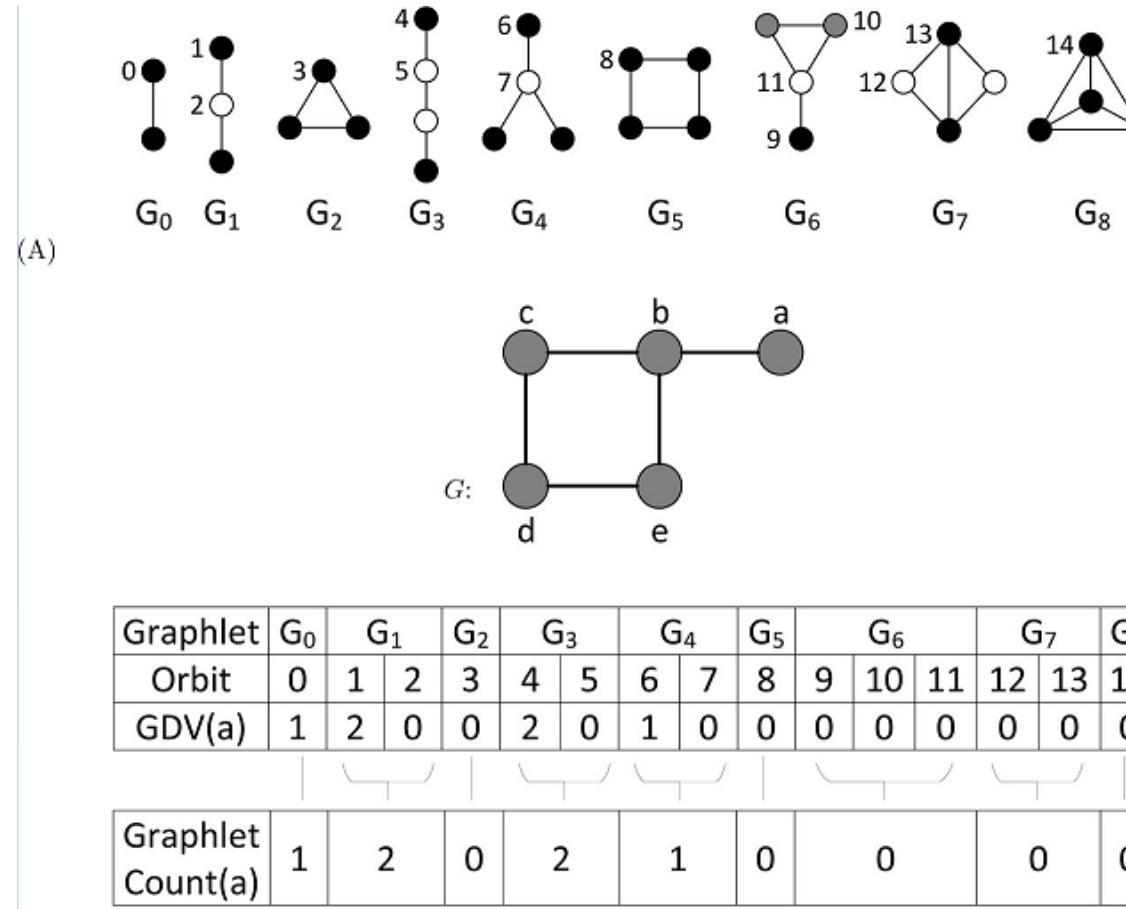
- Graphlets are **induced, non-isomorphic** subgraphs that **describe the structure of node u's network neighbourhood**



- Graphlet Degree Vector (GDV): A count vector of graphlets rooted at a given node
- Graphlet degree vector provides a measure of a node's local network topology (more detail than node degrees or clustering coefficient)



- Graphlets are induced, non-isomorphic subgraphs that describe the structure of node u's network neighbourhood
- Example:

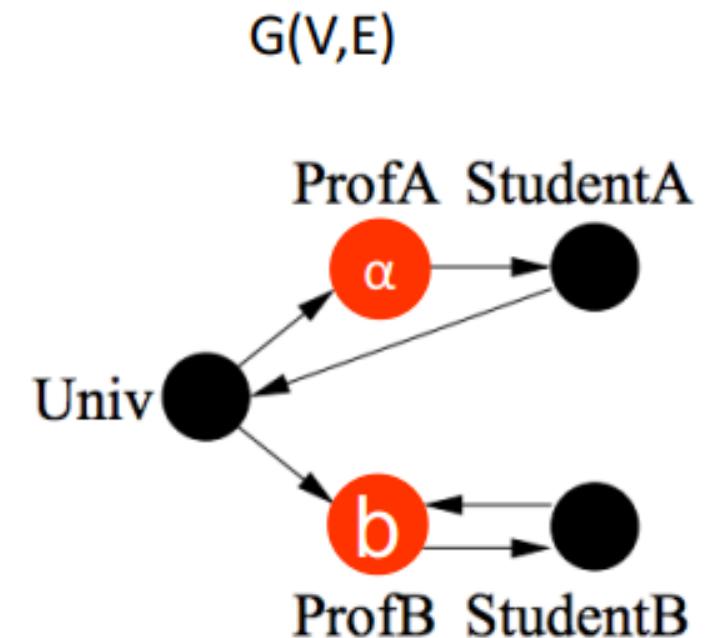


## ➤ Idea:

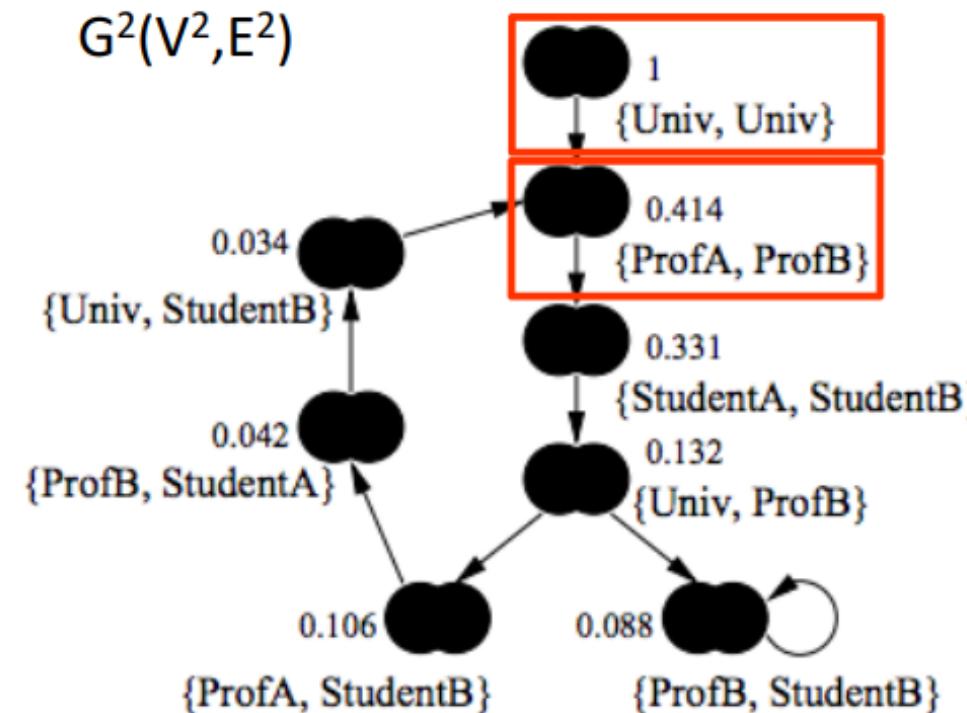
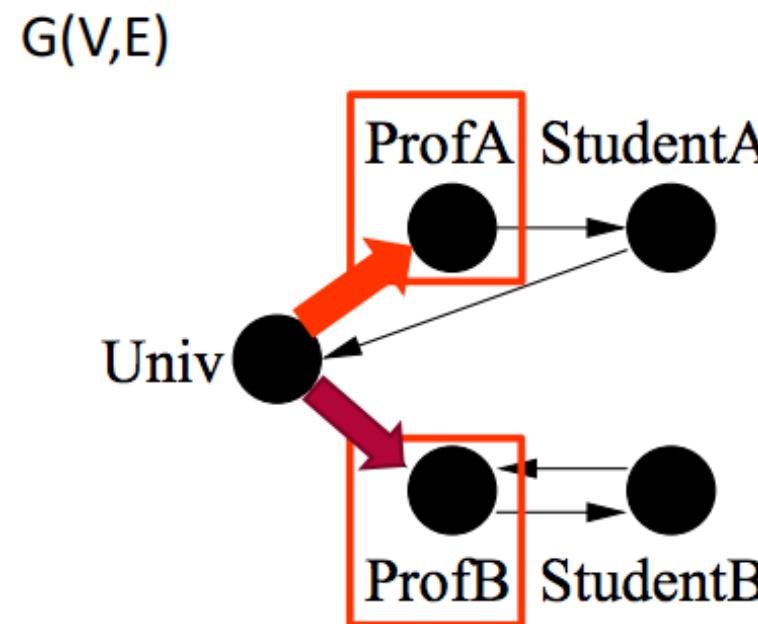
- Two objects are similar if they are **referenced by similar objects**

decay factor in  $[0,1]$

$$s(a, b) = \frac{\text{Avg similarity between in-neighbors of } a \text{ and in-neighbors of } b}{\text{total # of in-neighbors pairs}}$$



- Intuition: Computing SimRank is like propagating on the  $G^2$  graph of node-node pairs
  - The source of similarity is self-vertices, like (Univ, Univ)
  - Similarity propagates along pair-paths in  $G^2$ , away from the sourcescc



Structural context

- The keynote of method is to let **every point iteratively spread its label information** to its neighbours **until a global stable state** is achieved
  - Input:
    - Given a point set  $X = \{x_1, \dots, x_l, x_{l+1}, \dots, x_n\} \in R^m$
    - A label set  $L = \{1, \dots, c\}$ , the first  $l$  points  $x_i$  are labelled as  $y_i \in L$  and the remaining points  $x_u$  ( $l + 1 \leq u \leq n$ ) are unlabelled
  - Output:
    - Predict the label of the unlabelled points
- 

- Let  $F$  denote the set of  $n \times c$  matrices with nonnegative entries. A matrix  $F = [F_1, F_2, \dots, F_n]$  corresponds to a classification on the set  $X$  by labelling each point  $x_i$  as a label  $y_i = \text{argmax}_{\{j \leq c\}} F_{ij}$
- We can understand  $F$  as a vectorial function  $F: X \rightarrow R^c$  which assigns a vector  $F_i$  to each point  $x_i$

The algorithm is as follows:

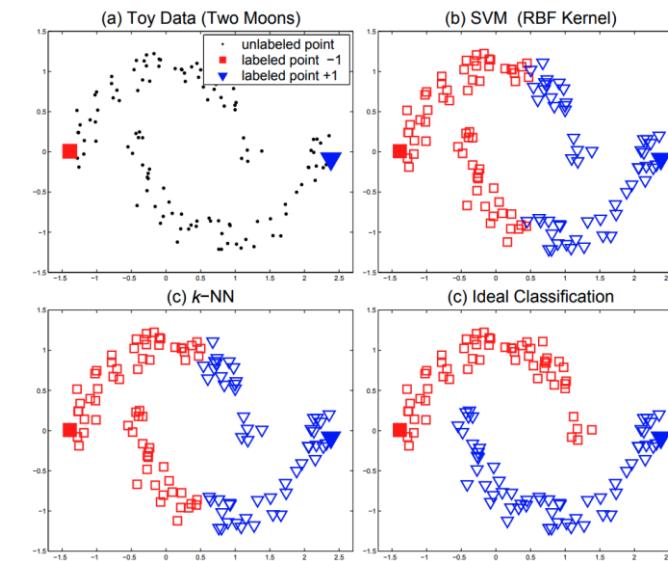
1. Form the affinity matrix  $W$  defined by:

$$W_{ij} = \begin{cases} e^{\frac{-\|x_i - x_j\|^2}{2\sigma^2}}, & \text{if } i \neq j \\ 0, & \text{if } i = j \end{cases}$$

2. Construct the matrix  $S = D^{-1/2}WD^{-1/2}$

where  $D$  is a diagonal matrix with its  $(i,i)$ - element equal to the **sum of the  $i$ -th row of  $W$**

3. Iterate  $F(t+1) = \alpha SF(t) + (1 - \alpha)Y$  until convergence, where  $\alpha \in (0,1)$
4. Finally, the label of each unlabelled point is set to be the class of which it has received most information during the iteration process



# Learning with Local and Global Consistency with SVM: Sample code 31

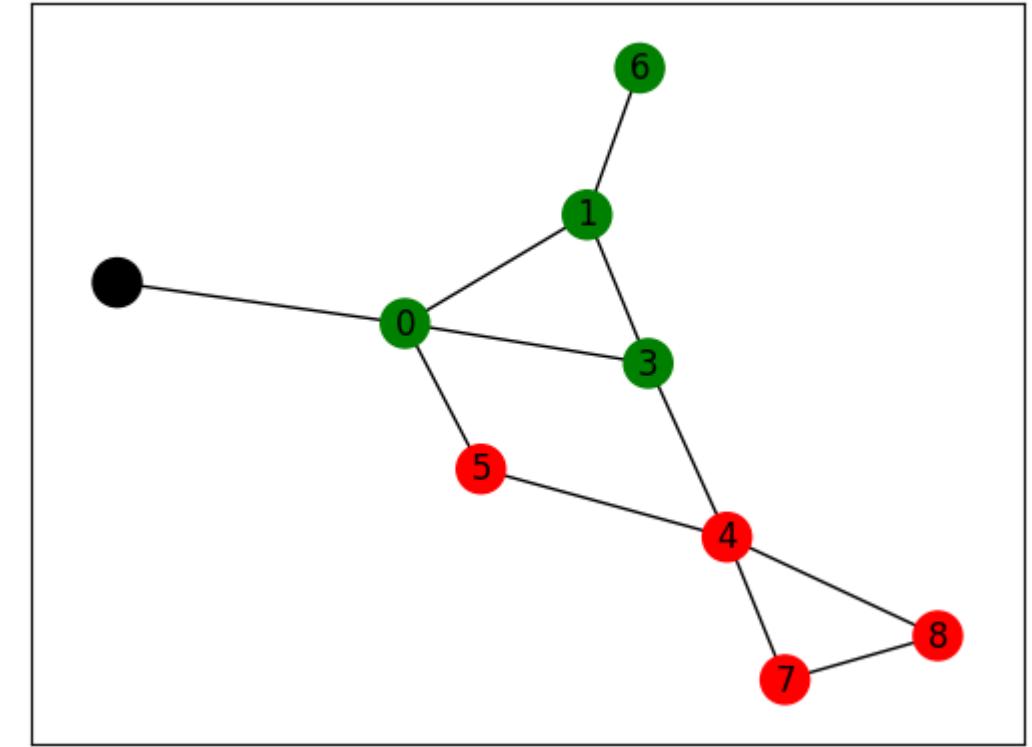
```
from networkx.algorithms import node_classification
import networkx as nx

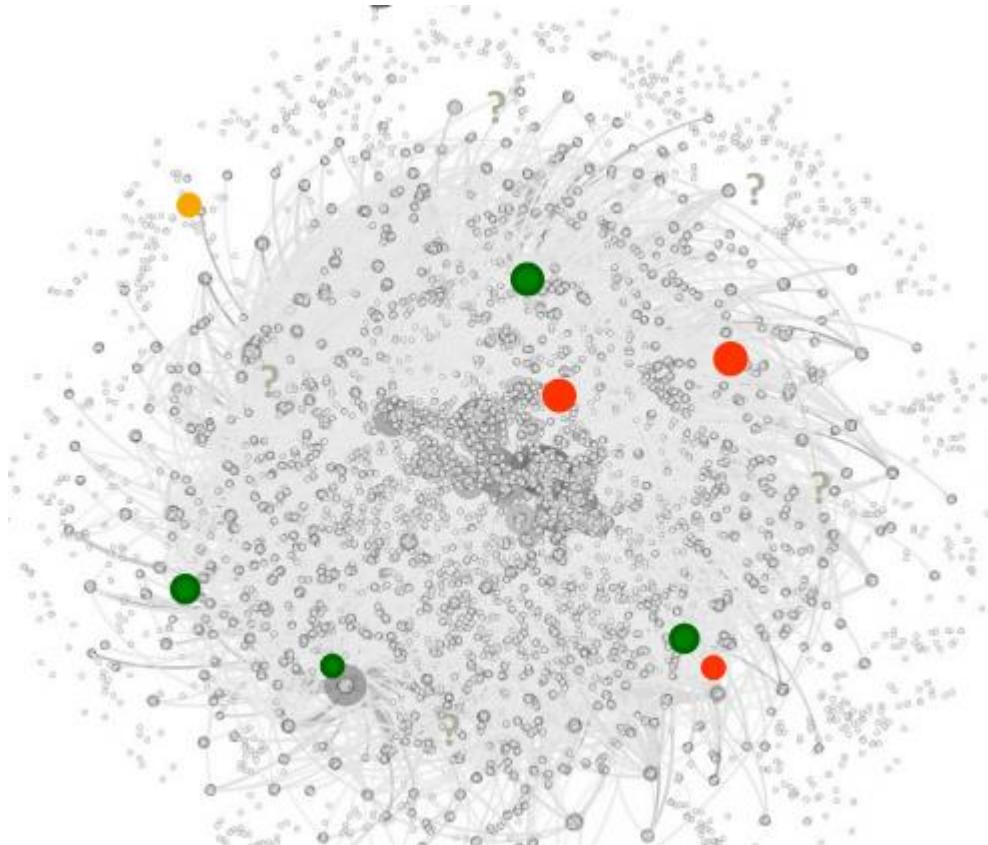
G = nx.Graph()
# add node/edge pairs
G.add_edges_from([(0, 1),(0, 2),(0, 3),(0, 5),(1, 3), (1, 6),(3, 4),(4, 5),(4, 7),(4,8),(7,8)])

G.nodes[0]['label'] = 'A'
G.nodes[1]['label'] = 'A'
G.nodes[2]['label'] = ''
G.nodes[3]['label'] = 'A'
G.nodes[4]['label'] = 'B'
G.nodes[5]['label'] = 'B'
G.nodes[6]['label'] = 'A'
G.nodes[7]['label'] = 'B'
G.nodes[8]['label'] = 'B'

predicted = node_classification.local_and_global_consistency(G)
predicted

['A', 'A', 'A', 'A', 'B', 'A', 'B', 'B', 'B']
```





➤ **Given:**

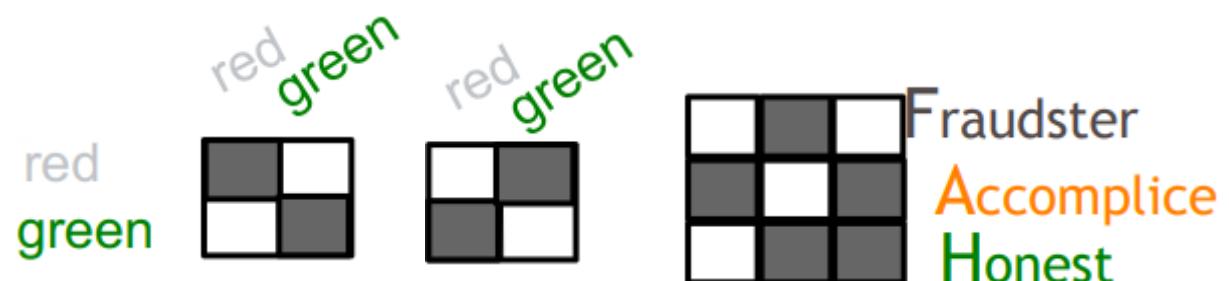
- Graph and few labelled nodes

➤ **Find:**

- class (red/green) for rest nodes

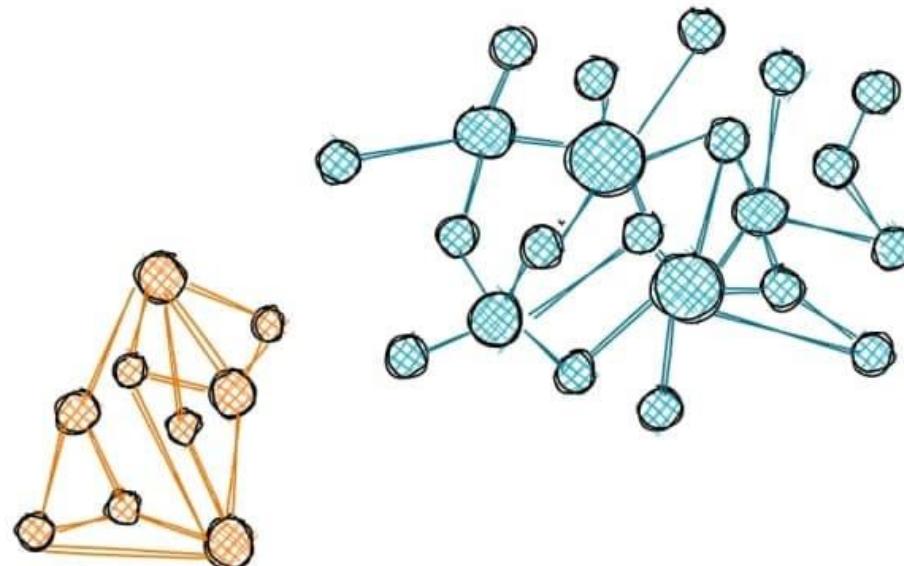
➤ **Assuming:**

- Network effect (homophily/heterophily)

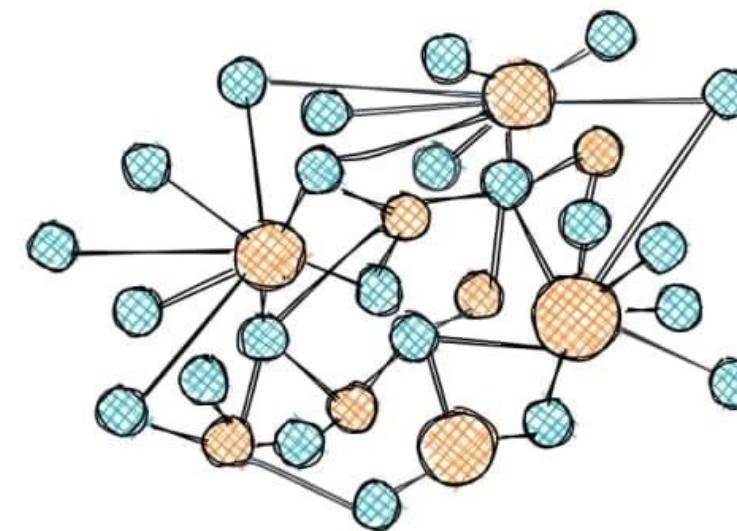


- **Homophily:** the tendency of individuals to associate and bond with similar others
- **Heterophily:** the tendency to interact with others of different type, while the connectivity within the groups is sparse

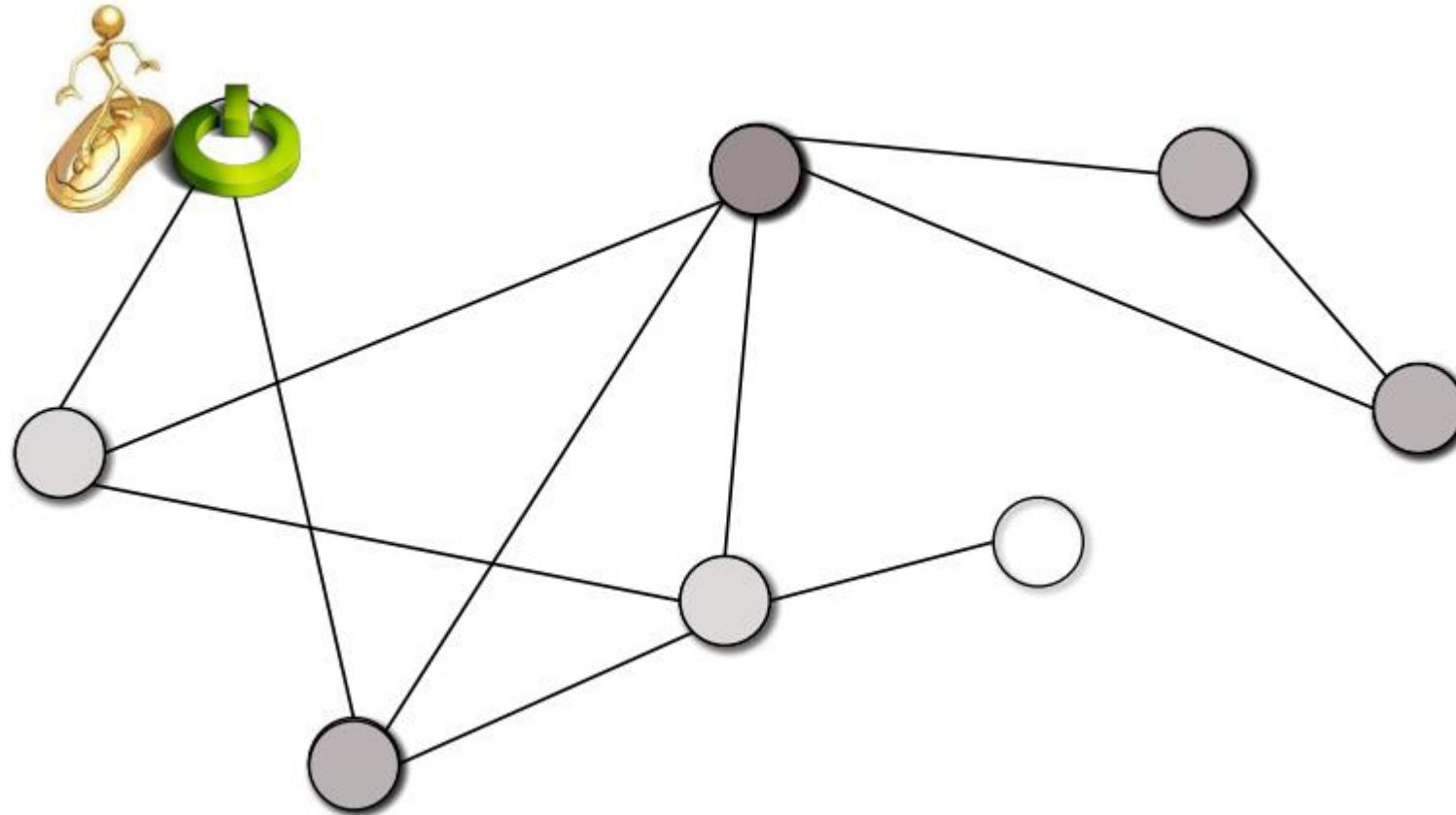
Homophily



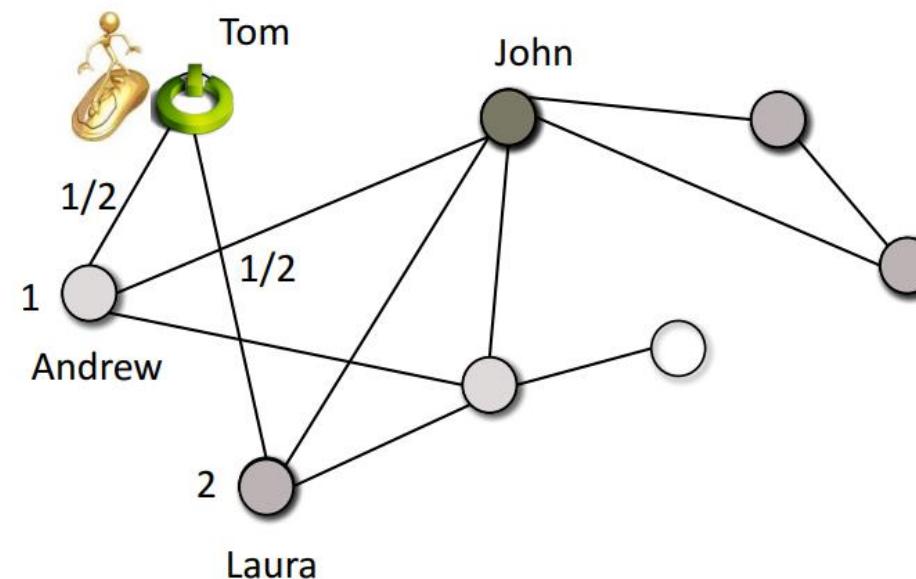
Heterophily



- **Idea:** Propagate labels from a set of nodes to the rest of the graph



- In a random walk, you assume that the **walker moves randomly** and **chooses one of the neighbours** to visit
- In the figure:
  - Tom chooses Andrew or Laura with probability  $1/2$
  - Once he chooses one it increases the number of times he visited that node
  - Continue the process until nothing changes anymore (at a probabilistic level)

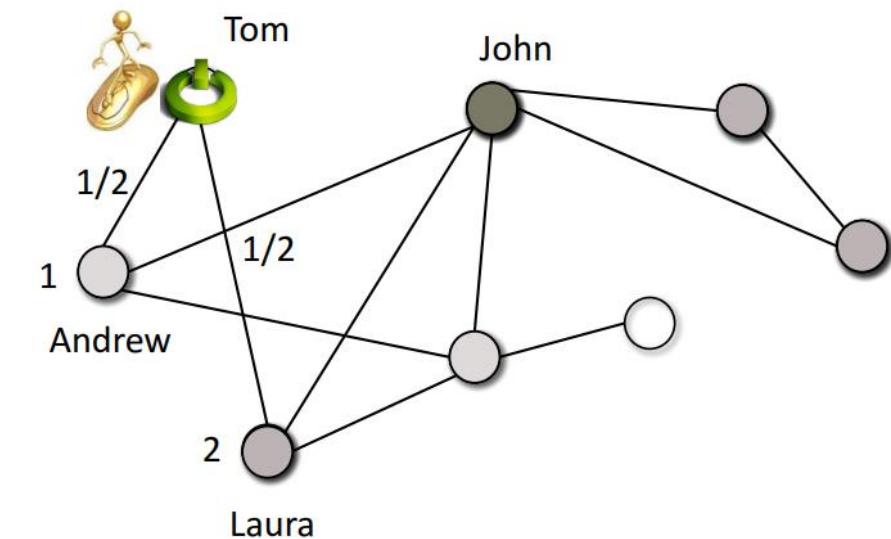


John will receive many visits since many nodes are connected to him

- Now assume that with probability  $c$  you perform another move and with probability  $(1-c)$  you jump back to Tom
- Therefore, the probability for the walker of being in Tom place will be:

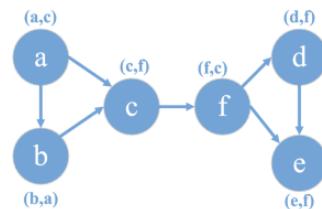
$$Tom(t) = c \frac{Prob(Andrew)(t-1) + Prob(Laura)(t-1)}{3} + (1 - c)$$

Probability of visiting Andrew at time (t-1)  
Number of Andrew and Laura's neighbors  
Probability of jumping back to Tom

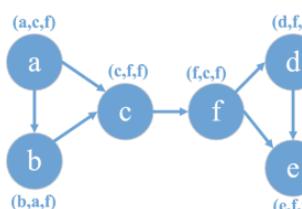


- At each step of the process, **each vertex updates its label to a new one which corresponds to the most frequent label among its neighbours**
- For each node  $v \in V$ ,  $v$  updates its label according to:

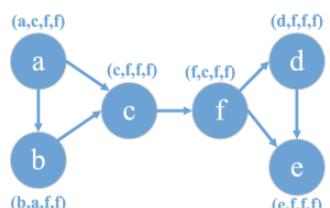
$$l_v = \arg \max \sum_{u \in N(v)} [l_u == l]$$



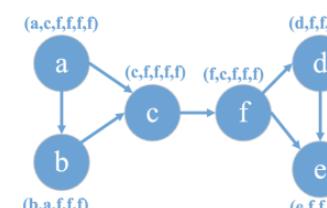
(a) Result after the first label propagation



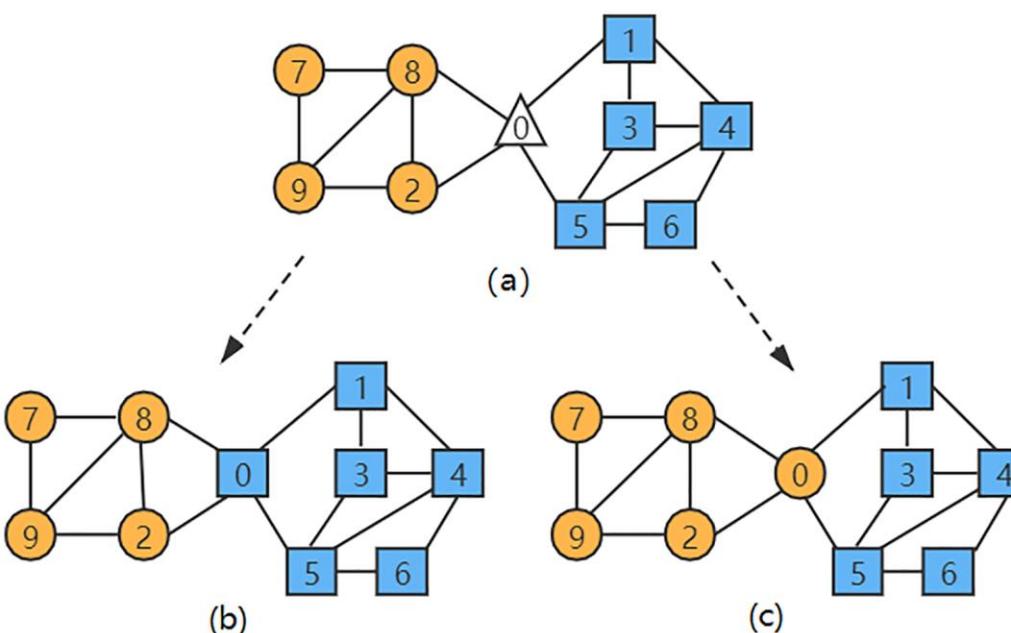
(b) Result after the second label propagation



(c) Result after the third label propagation



(d) Result after the fourth label propagation

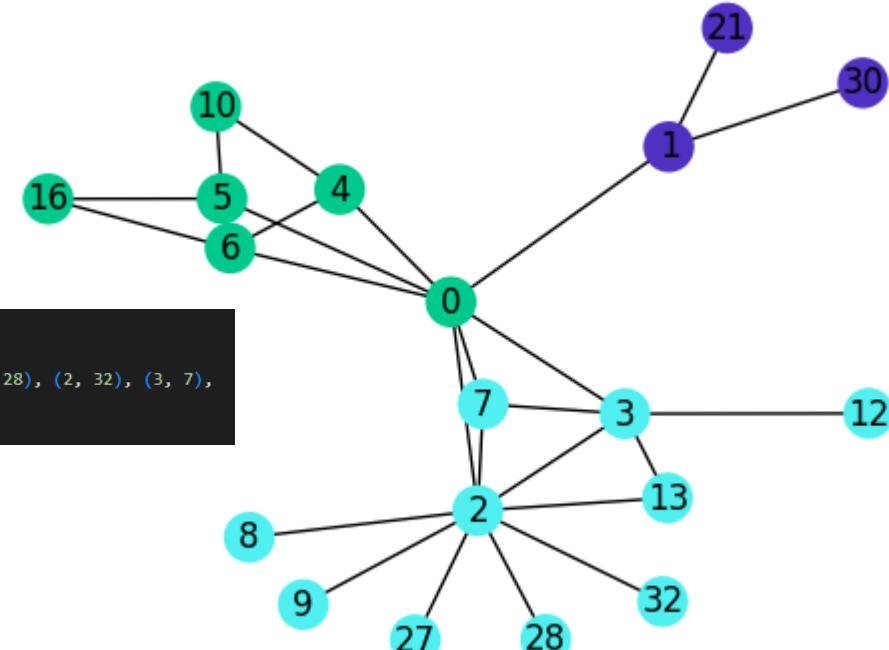


# Label propagation: Sample code

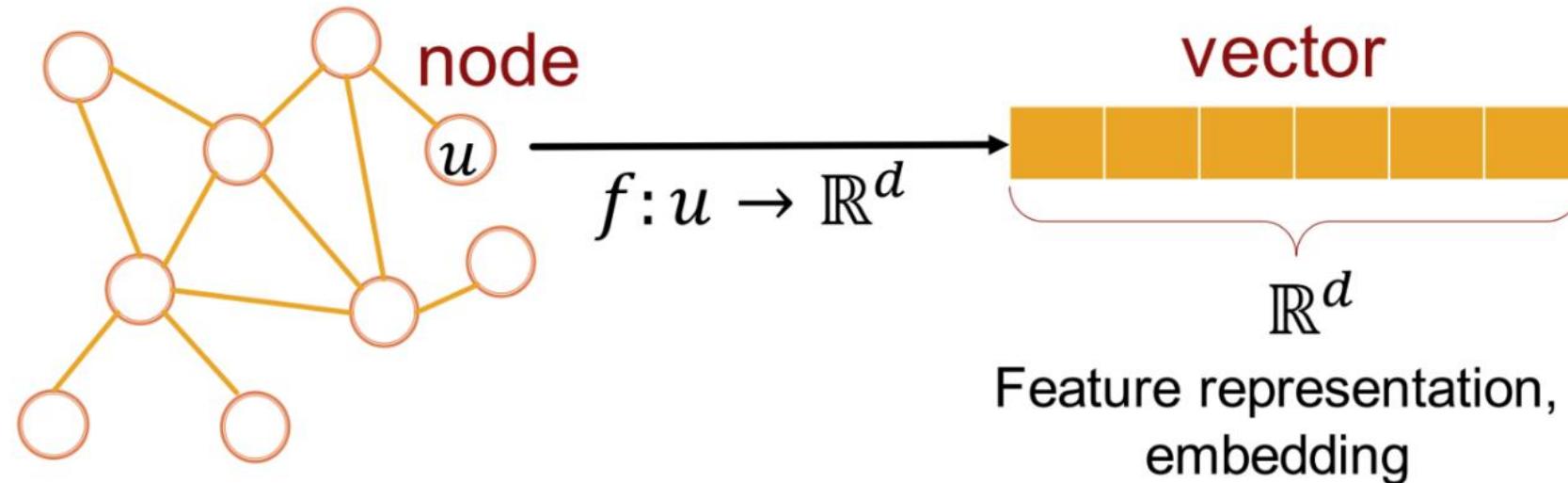
```
# Import libraries
import pandas as pd #For reading dataset files
import networkx as nx #For network creation/analysis
from networkx.algorithms import community
import community as community_louvain
import matplotlib.pyplot as plt #For plotting graphs
import igraph as ig
%matplotlib inline
```

```
G = nx.Graph()
# add node/edge pairs
G.add_edges_from([(0, 1), (0, 2), (0, 3), (0, 4), (0, 5), (0, 6), (0, 7), (1, 21), (1, 30), (2, 3), (2, 7), (2, 8), (2, 9), (2, 13), (2, 27), (2, 28), (2, 32), (3, 7),
(3, 12), (3, 13), (4, 6), (4, 10), (5, 6), (5, 10), (5, 16), (6, 16)])
```

```
colors = ["#00C98D", "#5030C0", "#50F0F0"]
pos = nx.spring_layout(G)
lst_m = community.label_propagation_communities(G)
color_map_b = {}
keys = G.nodes()
values = "black"
for i in keys:
    color_map_b[i] = values
counter = 0
for c in lst_m:
    for n in c:
        color_map_b[n] = colors[counter]
    counter = counter + 1
nx.draw_networkx_edges(G, pos)
nx.draw_networkx_nodes(G, pos, node_color=dict(color_map_b).values())
nx.draw_networkx_labels(G, pos)
plt.axis("off")
plt.show()
```



- Graph Representation Learning:
  - Learn efficient task-independent feature for machine learning with graphs  
→ Map nodes into an embedding space
- Similarity of embeddings between nodes indicates their similarity in the network
- For simplicity, no node features or extra information is used

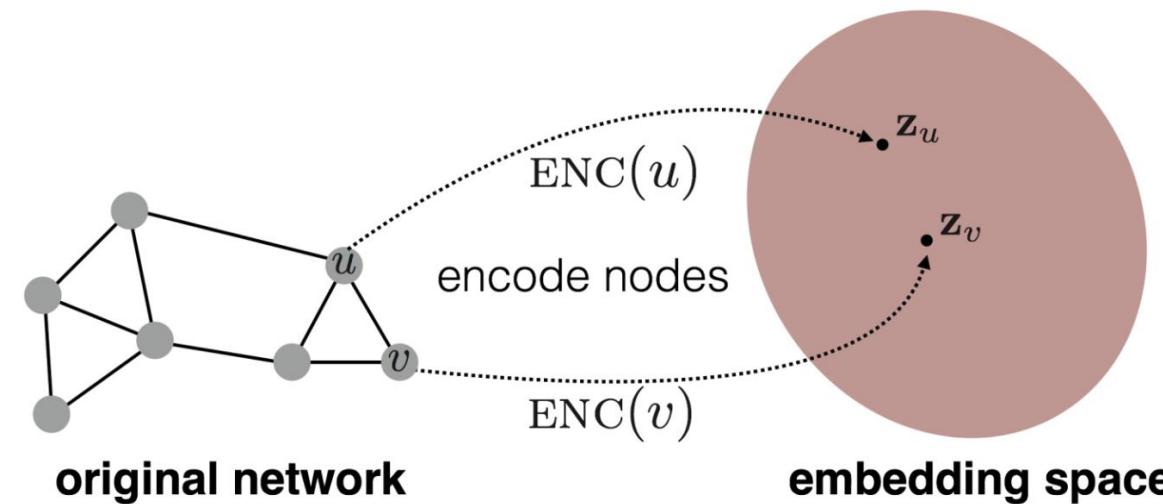


- Goal: Encode nodes → similarity in embedding space (dot product)  $\approx$  similarity in the original graph

$$\text{similarity}(u, v) \approx \mathbf{z}_v^T \mathbf{z}_u$$

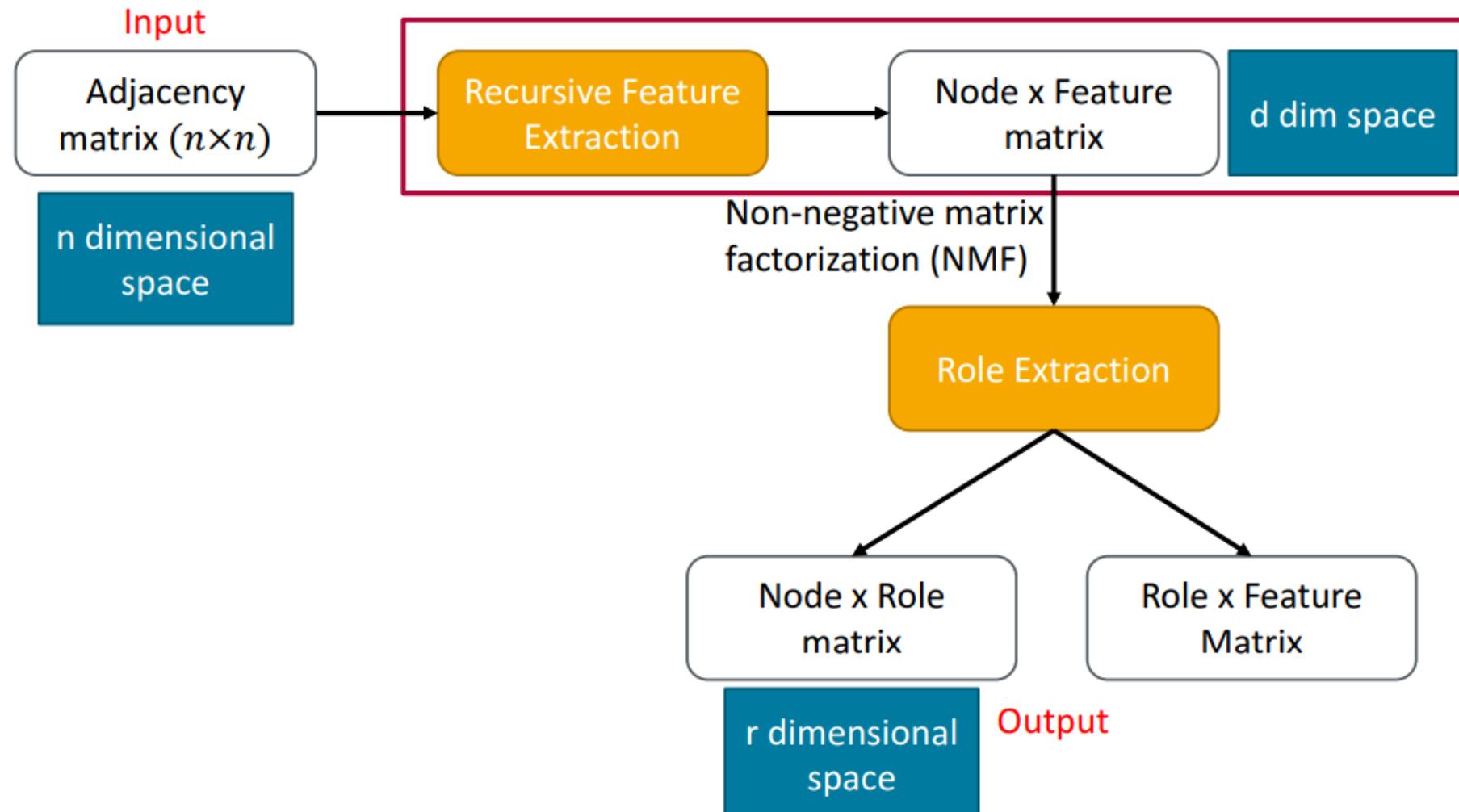
in the original network                                  Similarity of the embedding

We need to define:  
 $\text{ENC}(u)$   
 $\text{Similarity}(u, v)$



# RoIX: Role eXtraction algorithm

- Takes features extracted with **ReFeX** and **factorizes the binary node-feature matrix** in order to create low dimensional structural node representations



# RoIX: Role eXtraction algorithm: Sample code

42

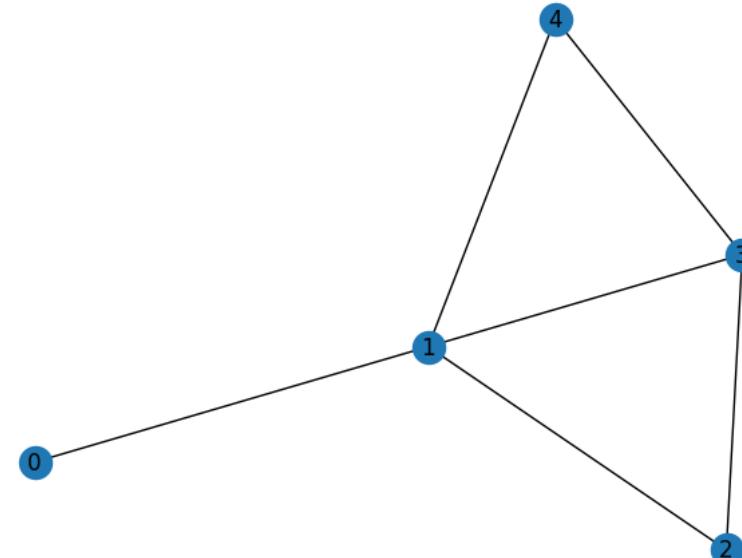
```
# assign node roles
role_extractor = RoleExtractor(n_roles=None)
role_extractor.extract_role_factors(features)
node_roles = role_extractor.roles

print('\nNode role assignments:')
pprint(node_roles)

print('\nNode role membership by percentage:')
print(role_extractor.role_percentage.round(2))

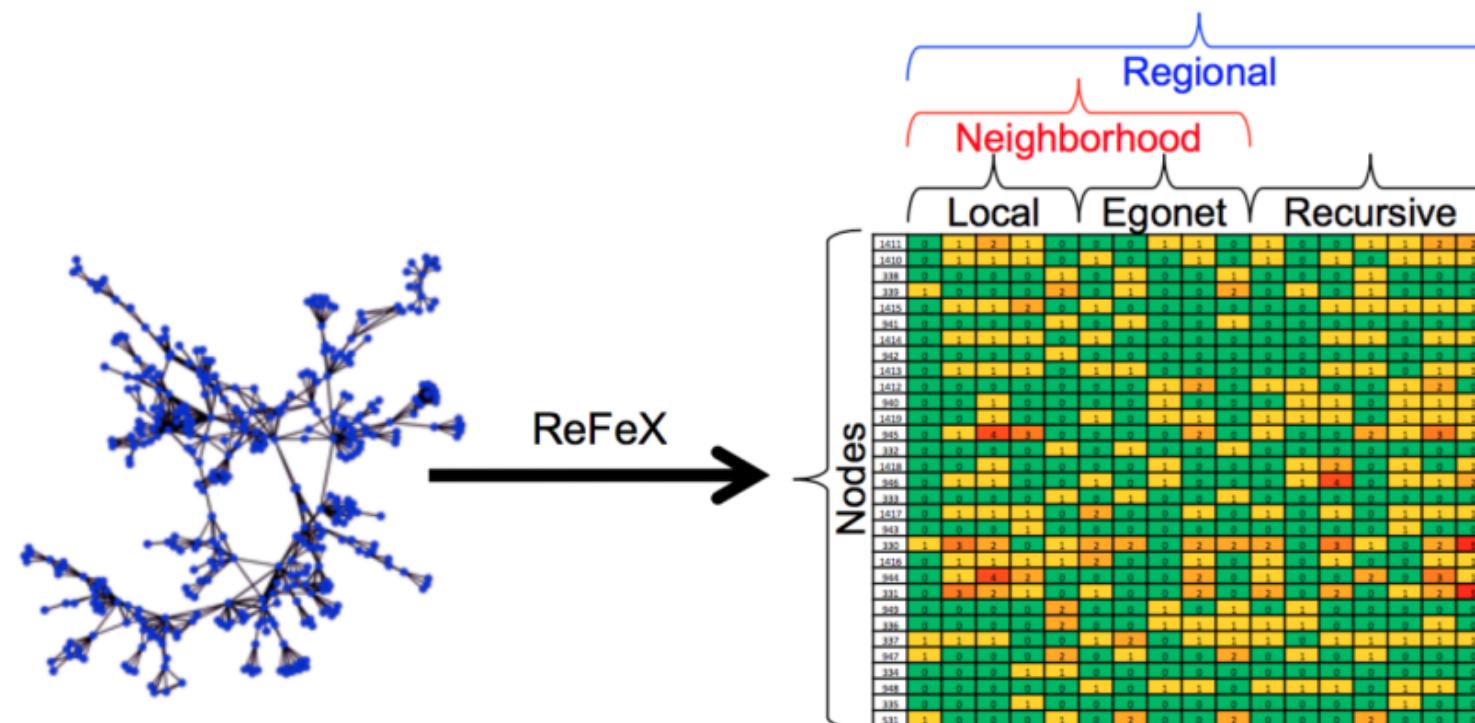
Node role assignments:
{0: 'role_1', 1: 'role_0', 2: 'role_1', 3: 'role_0', 4: 'role_1'}

Node role membership by percentage:
  role_0  role_1
0    0.03    0.97
1    0.97    0.03
2    0.25    0.75
3    0.69    0.31
4    0.25    0.75
```



# Recursive Feature extraction (ReFex)

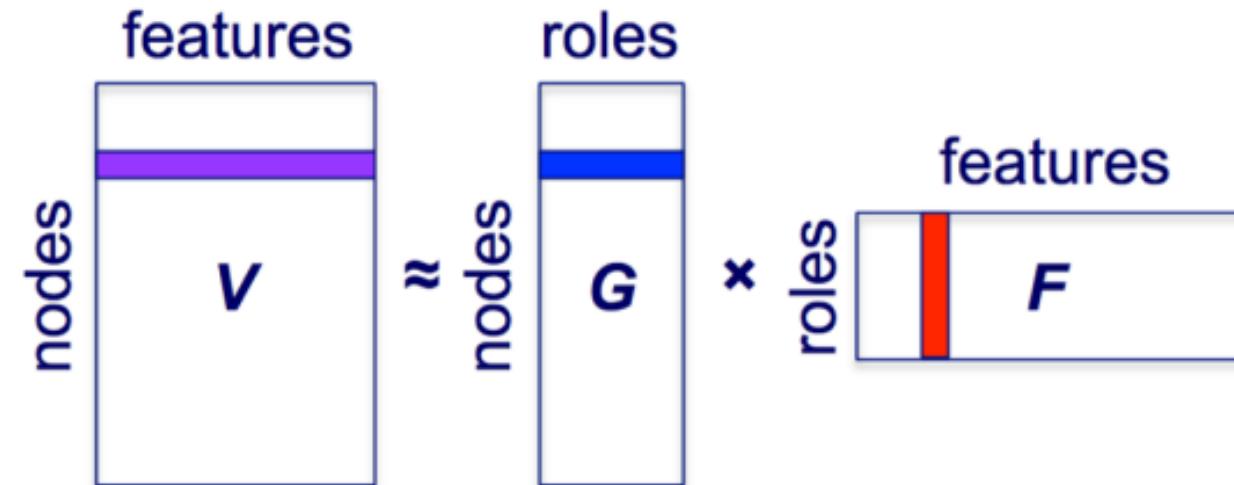
- Transform the network **connectivity** into **recursive structural features**
- Technically, embeds the graph into an  $|\mathcal{F}|$  dimensional space, where  $\mathcal{F}$  is a set of **features** (degree, self-loops, avg. edge weight, # of edges in egonet)



- Local:
  - Measures of the node **degree**
- Egonet:
  - The **egonet** (or ego-network) of a node is the node itself, the adjacent nodes, and the graph induced by those nodes
  - Computed based on each node's ego network: #of within-egonet edges, #of edges entering & leaving the egonet
- Recursive
  - Some **aggregate** (mean, sum, max, min, etc.) of another feature over a node's neighbours
  - The aggregation can be computed over any real-valued feature, including other recursive features

## Role extraction: Feature grouping

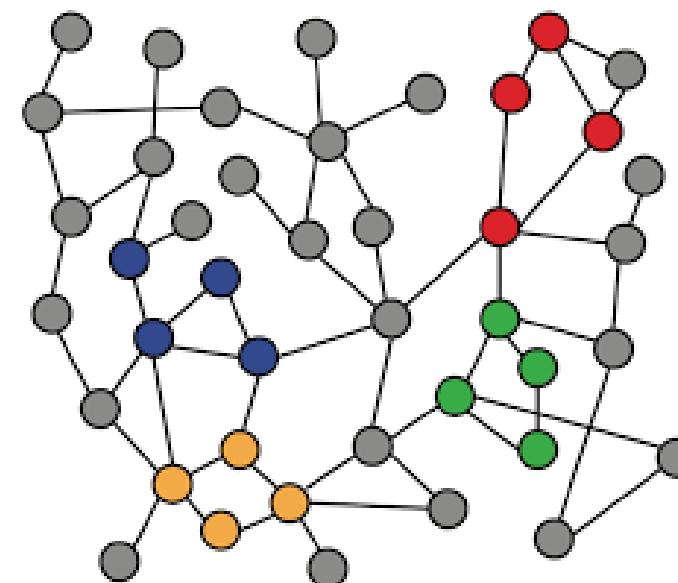
- Find  $r$  overlapping clusters in the feature space
  - Each node can have multiple roles at the same time.
- Generate a rank  $r$  approximation of the node  $\times$  feature matrix  $V$
- Use **non-negative matrix factorization**:  $V \approx GF$



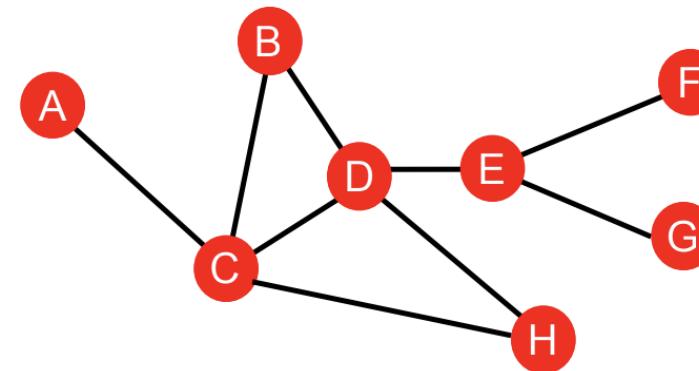
- The  $G$  matrix assigns nodes to roles
- The  $F$  matrix represents how the features explain the roles

- **Local** features: capture the importance of a node in a graph
  - Node degree: Simply counts the number of neighboring nodes
  - Node centrality: Model **importance of neighboring nodes** in a graph
    - Different modeling choices: eigenvector centrality, betweenness centrality, closeness centrality, Katz centrality, Pagerank
  - Clustering coefficient: Measures how connected neighboring nodes are
  - Neighboring information
- **Global** features: Capture topological properties of local neighborhood around a node
  - Graphlet: Counts the occurrences of different graphlets
  - Simrank: Structural-Context Similarity
  - Label Propagation
  - Node embedding

- **Goal:** Characterize the structure and connectivity of nodes and edges in the network:
  - Distance-based Features
  - Local Neighbourhood Overlap Feature
  - Global Neighbourhood Overlap Feature



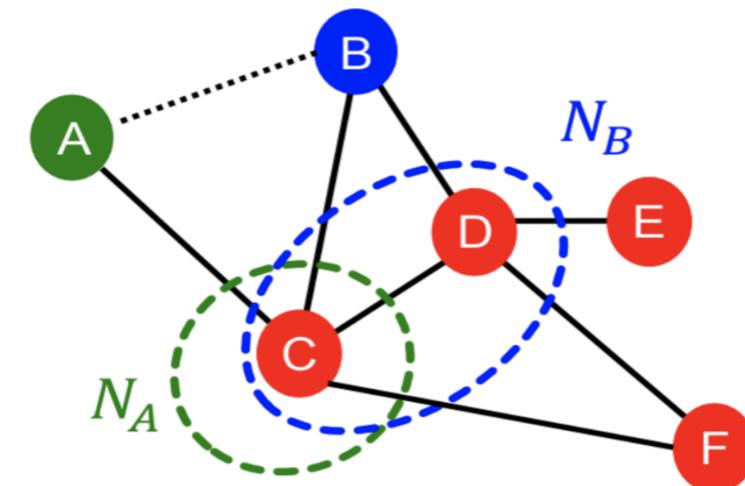
- Shortest-path distance between two nodes
- For example:



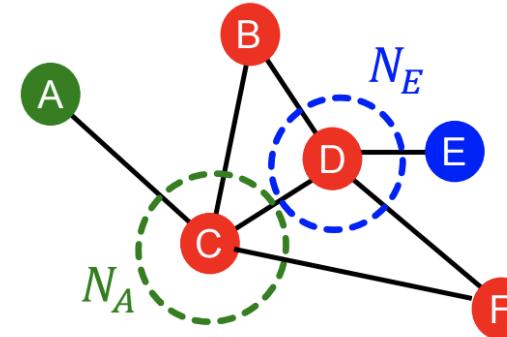
$$\begin{aligned}S_{BH} &= S_{BE} = S_{AB} = 2 \\S_{BG} &= S_{BF} = 3\end{aligned}$$

- However, this does not capture the degree of neighbourhood overlap:
- Node pair (B, H) has 2 shared neighbouring nodes, while pairs (B, E) and (A, B) only have 1 such node

- Captures # neighbouring nodes shared between two nodes  $v_1$  and  $v_2$ :
- Common neighbours:  $|N(v_1) \cap N(v_2)|$ 
  - 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:  $\frac{1}{\log(k_C)} = \frac{1}{\log 4}$



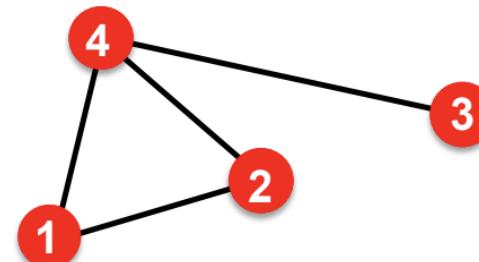
- Limitation of local neighbourhood overlapping features:
  - Metric is always zero if the two nodes do not have any neighbours in common
  - However, the two nodes may still potentially be connected in the future



$$\begin{aligned}N_A \cap N_E &= \emptyset \\|N_A \cap N_E| &= 0\end{aligned}$$

→ Global neighbourhood overlap features resolve the limitation by considering the entire graph

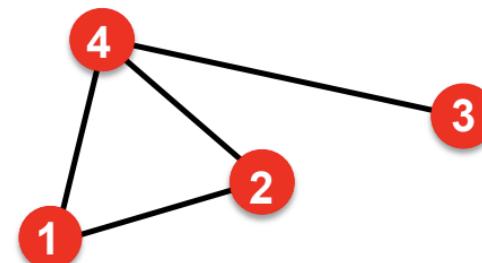
- Katz index:
  - Count the number of walks of all lengths between a given pair of nodes.
- How to compute number of walks?
  - Powers of the graph adjacency matrix



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

$A_{i,j} = 1$  if node  $i, j$  are connected

- So, what is number of walks?
- We want to show:  $P^{(K)} = A^k$
- where  $P_{uv}^{(K)} = \# \text{walks of length } K \text{ between } u \text{ and } v$
- $P_{uv}^{(1)} = \# \text{walks of length 1 (direct neighborhood) between } u \text{ and } v = A_{uv}$



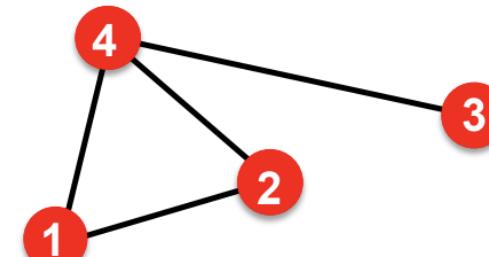
$$P_{12}^{(1)} = A_{12}$$

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

$A_{i,j} = 1$  if node  $i, j$  are connected

## ■ How to compute $P_{uv}^{(2)}$ ?

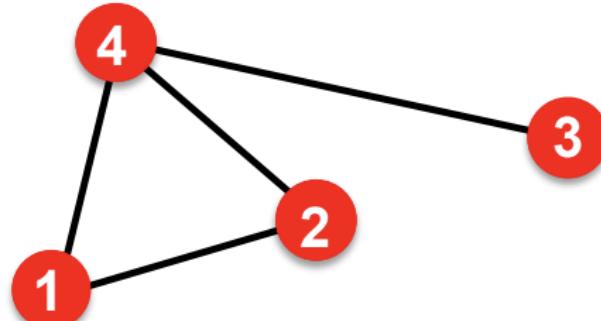
- Step 1: Compute #walks of length 1 between each of  $u$ 's neighbor and  $v$
- Step 2: Sum up these #walks across  $u$ 's neighbors
- $P_{uv}^{(2)} = \sum_i A_{ui} * P_{iv}^{(1)} = \sum_i A_{ui} * A_{iv} = A_{uv}^2$



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

$$\begin{aligned}
 A^2 &= \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \times \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 2 & 1 & 1 & 1 \\ 1 & 2 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 3 \end{pmatrix} \\
 &\text{Power of} \\
 &\text{adjacency}
 \end{aligned}$$

- Using powers of adjacency matrix, we can calculate number of walks of all lengths between a pair of nodes
- $A_{uv}$  specifies #walks of length 1 (direct neighbourhood) between u and v
- $A_{uv}^2$  specifies #walks of length 2 (neighbour of neighbour) between u and v
- $A_{uv}^l$  specifies #walks of length l



- **Katz index** between  $v_1$  and  $v_2$  is calculated as **sum over all walk lengths**

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

#walks of length  $l$   
between  $v_1$  and  $v_2$

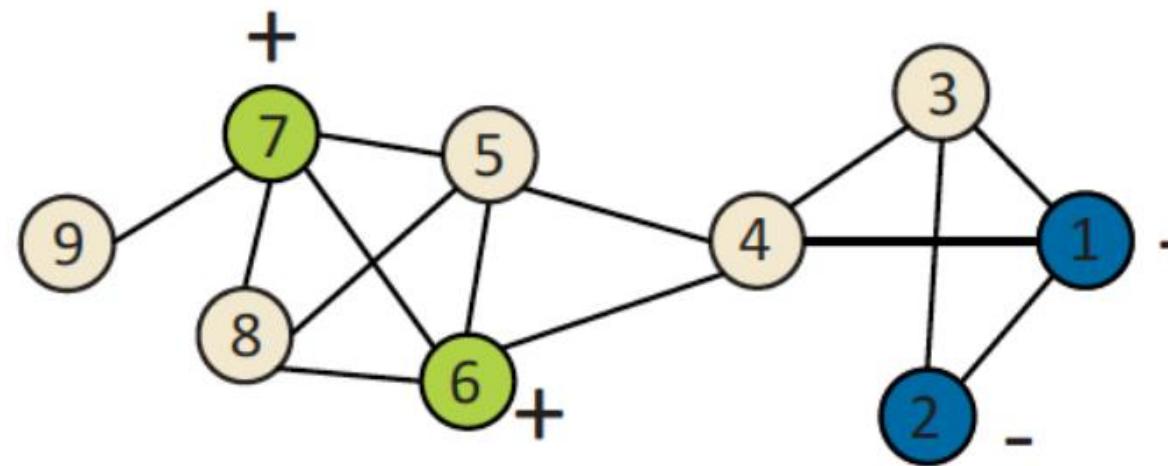
$0 < \beta < 1$ : discount factor

- **Katz index matrix** is computed in closed-form:

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

- Distance-based features:
  - Calculate shortest path between 2 nodes but cannot capture overlapping neighbours
- Local neighbourhood overlap:
  - Capture number of sharing neighbourhoods between 2 nodes
  - Only focus nodes within 2-hop
- Global neighbourhood overlap:
  - Katz index use entire graph structure to score 2 nodes
  - It can capture the structure globally

- Given a graph and few nodes for which we know the “**label**” or a “**class**,” how can we predict user attributes or interests?



Predict the labels for non-marked nodes?

- Is this a friend or an acquaintance?
- Recommendation systems to suggest objects (music, movies, activities)
- Automatically understand roles in a network (hubs, activators, influencing nodes, etc.)
- Identify experts for question answering systems
- Targeted advertising
- Study of communities (key individuals, group starters,...)
- Study of diseases and cures
- Identify unusual behaviours or behavioural changes
- Finding similar nodes and outliers

- Not all the nodes have labels
  - Users are not willing to provide explanations
- Roles are **not explicitly** declared
  - Who is more important in a company? (Think about the exchanged emails)
- Labels provided by the users can be misleading
- Labels are sparse
  - Some categories might be missing or incomplete

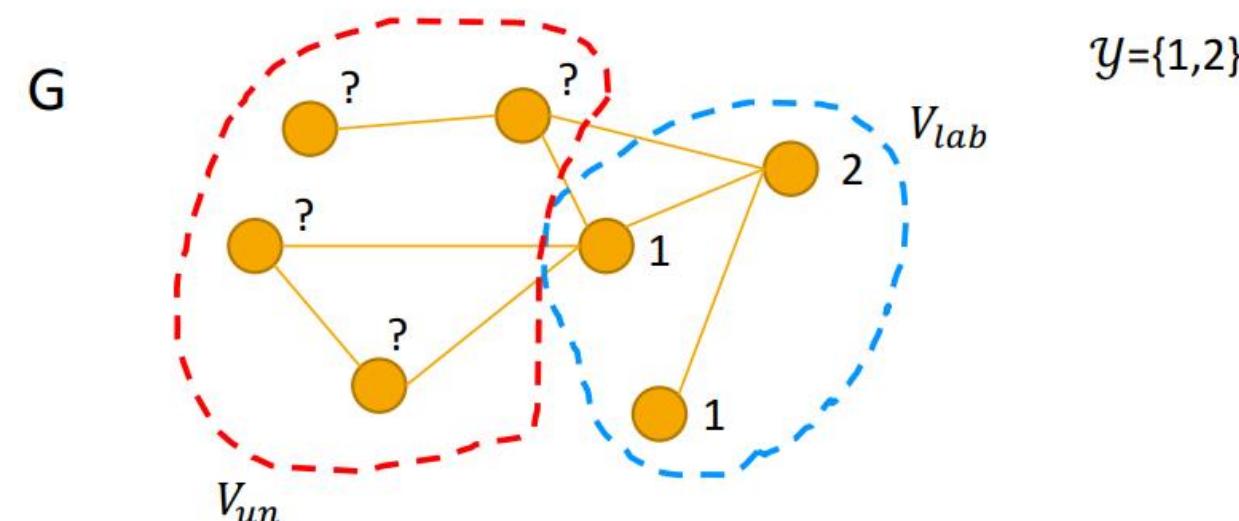
# Node classification problem

➤ Given:

- Graph  $G = (V, E, W)$  with vertices  $V$ , edges  $E$  and weight matrix  $W$
- Labeled nodes  $V_{lab} \subset V$ , unlabeled nodes  $V_{un} = V \setminus V_{lab}$
- $Y$  the set of  $m$  possible labels (e.g.,  $Y = \{\text{republican}, \text{democrat}\}$ )
- $Y_{lab} = \{y_1, y_2, \dots, y_l\}$  the labels on labeled nodes in  $V_{lab}$

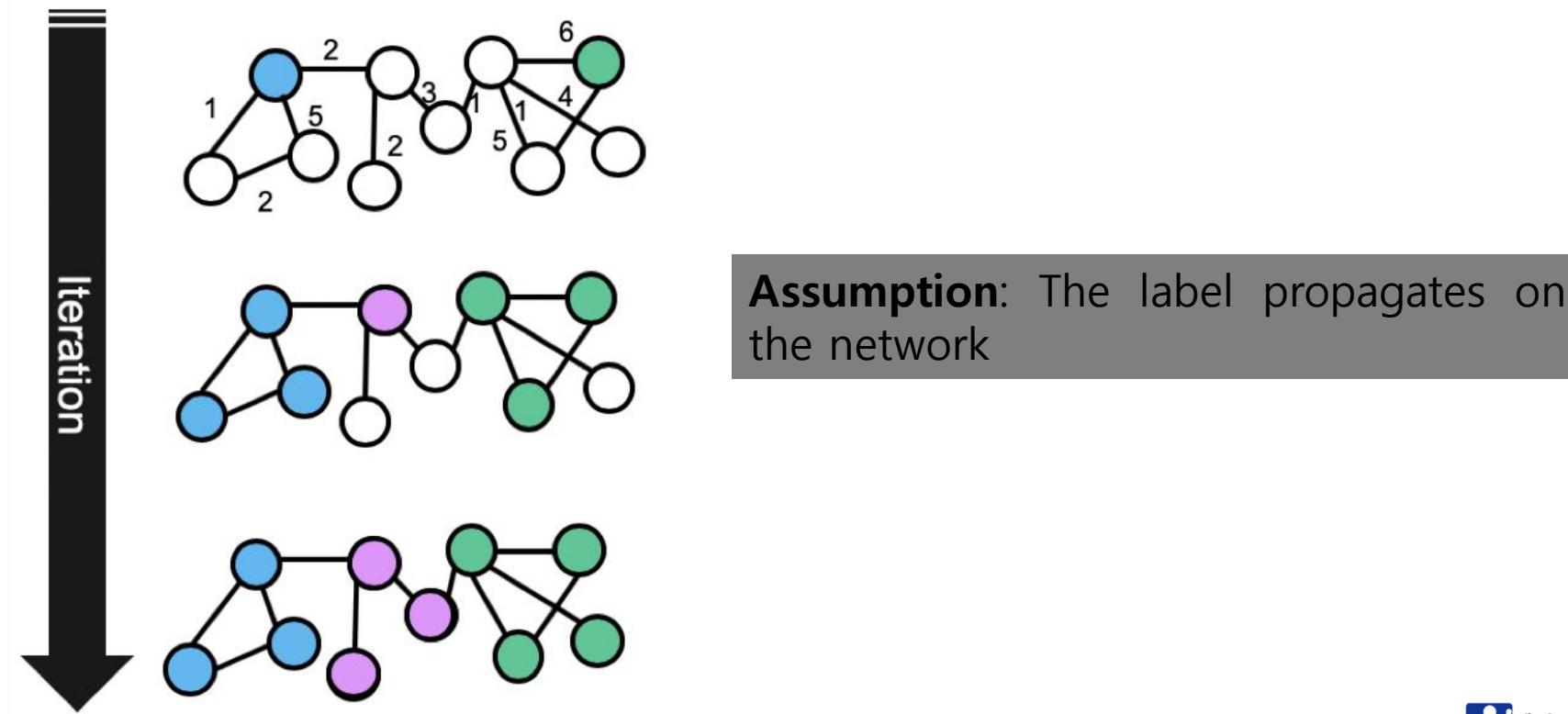
➤ Problem:

- Infer labels  $Y_{un}$  for all nodes in  $V_{un}$



- Can be generalized to multilabel and multiclass classification:
  - With multiclass classification assume that each labelled node has a **probability distribution on the labels**
- Can work on generalized graph structures
  - hypergraphs, graphs with weighted, labelled, timestamped edges, multigraphs, probabilistic graphs and so on

- The **graph structure** encodes important **information** for node classification
- So, it is reasonable to think that:
  - labels propagate in the network **following the links**
- Methods that work with points in the space perform poorly in a graph



- Node features:
  - Measurable characteristics of the nodes that help
    - Discriminating a node from another
    - Or stating the similarity with other nodes
- Examples of features:
  - In/out degree of the node
  - Number of L-labelled edges from that node
  - Number of paths in that goes through the node
  - Number of triangles
  - Degree and number within ego-net edges
  - etc

## ➤ **Similarity-based**

- Find nodes that **share the same characteristics** with other nodes

## ➤ **Iterative Learning**

- Learn a set of labels and **propagate the information to similar nodes**

## ➤ **Label Propagation**

- Labelled nodes **propagate the information to the neighbours** with some probability

# Similarity based

## ➤ Real-world Applications:

**Customers Who Bought This Item Also Bought**

The screenshot shows a section titled "Customers Who Bought This Item Also Bought" on an Amazon product page. It displays four books with their titles, authors, ratings, and prices. The books are:

- An Introduction to Statistical Learning: Gareth James (4.5 stars, #1 Best Seller)
- Advanced R (Hadley Wickham)
- Statistical Analysis of Network Data with R (Eric D. Kolaczyk)
- Machine Learning with R (Brett Lantz)

Each book entry includes a "Look Inside" button, a small thumbnail image, and a "Prime" delivery badge.



Scholar articles

On power-law relationships of the internet topology

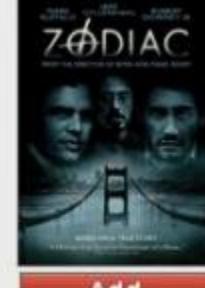
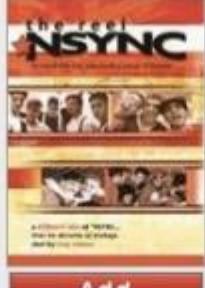
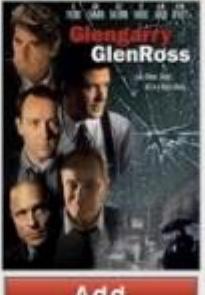
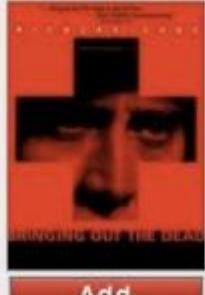
M Faloutsos, P Faloutsos, C Faloutsos - ACM SIGCOMM Computer Communication Review, 1999

Cited by 5151 - [Related articles](#) - All 88 versions

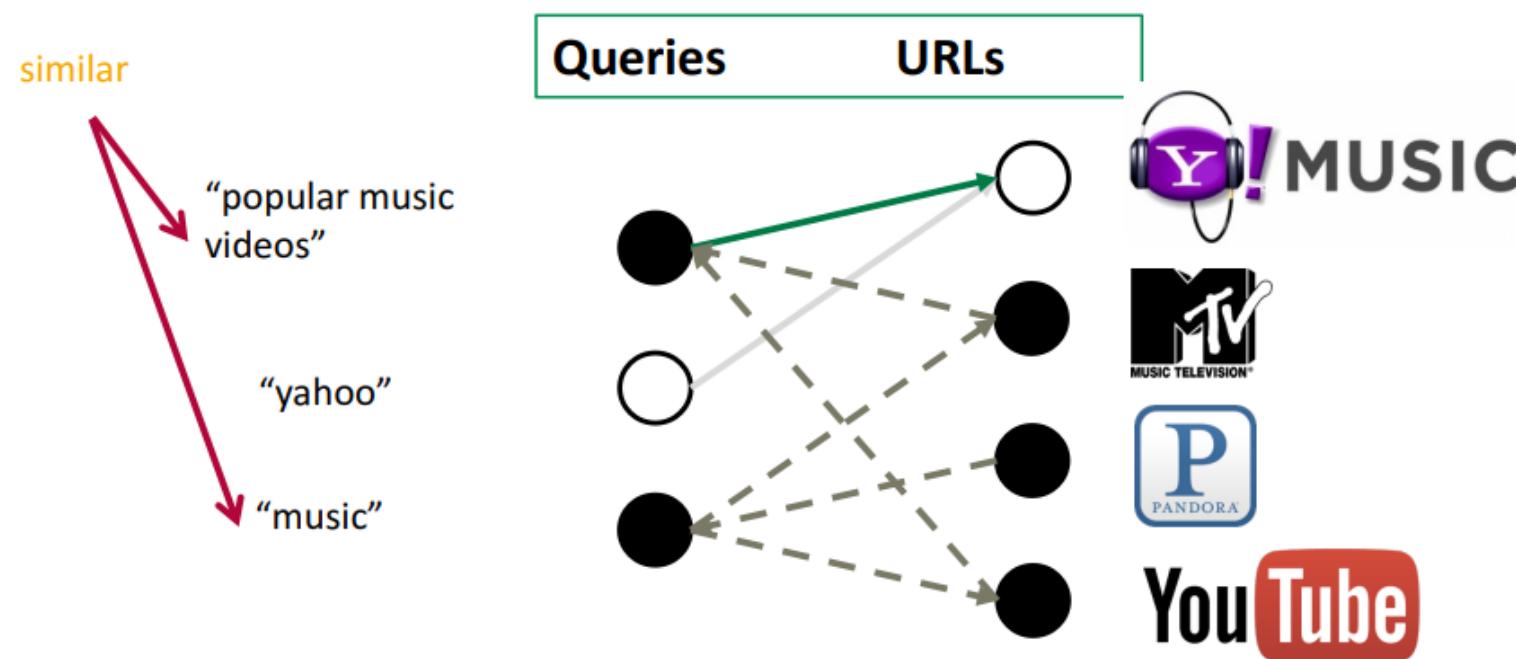
# Similarity based

## ➤ Movies recommendations

**More like The Social Network**

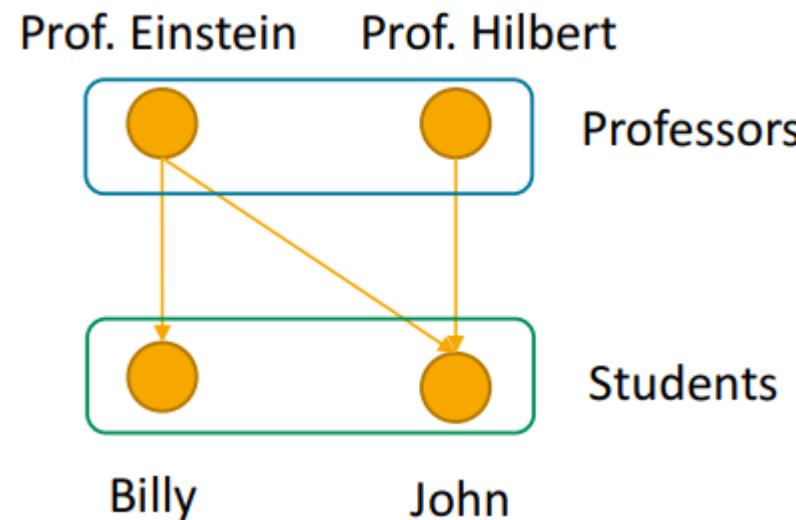
 <p><b>The Social Network</b> has been added to the Saved section of your DVD Queue  Availability: DVD: Unknown</p>	<p><b>Zodiac</b></p>  <p><b>'N Sync: The Reel 'N Sync</b></p>  <p><b>Edison Force</b></p> 	
<p><b>Parks and Recreation:</b> Season 1</p>  <p><b>Glengarry Glen Ross</b></p> 	<p><b>What's Eating Gilbert Grape</b></p>  <p><b>Bringing Out the Dead</b></p> 	<p><b>The Prize Winner of Defiance, Ohio</b></p> 
<p>Add</p> <p>★★★★★ <input type="radio"/> Not Interested</p>	<p>Add</p> <p>★★★★ <input type="radio"/> Not Interested</p>	<p>Add</p> <p>★★★★ <input type="radio"/> Not Interested</p>
<p>Add</p> <p>★★★★★ <input type="radio"/> Not Interested</p>	<p>Play Add</p> <p>★★★★★ <input type="radio"/> Not Interested</p>	<p>Add</p> <p>★★★★★ <input type="radio"/> Not Interested</p>

- Topical search engine is an engine that focuses on a particular topic
- It covers a part of the whole Web rather than a particular website - this is possible because Programmable Search Engine allows you to include multiple websites in the same engine



- Two nodes  $u$  and  $v$  are **structurally equivalent** if they have the same relationships to all other nodes
- Two nodes  $u$  and  $v$  are **automorphically equivalent** if all the nodes can be relabelled to form an isomorphic graph with the labels of  $u$  and  $v$  interchanged (just change the node id)
- Two nodes  $u$  and  $v$  are **regularly equivalent** if they are equally related to equivalent others

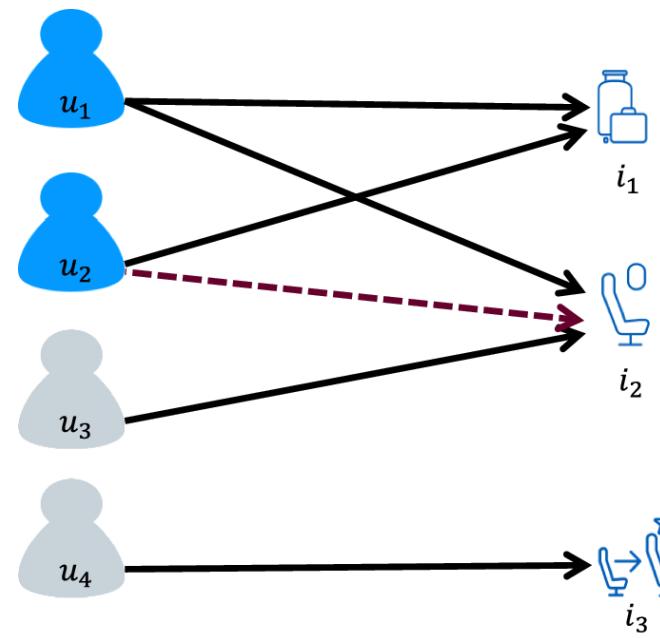
- Two nodes  $u$  and  $v$  are regularly equivalent if they are **equally related to equivalent others**
- Assumes a similarity between sets of nodes



Billy and John are similar because they are **both connected to a professor**  
Same for prof. Einstein and Hilbert

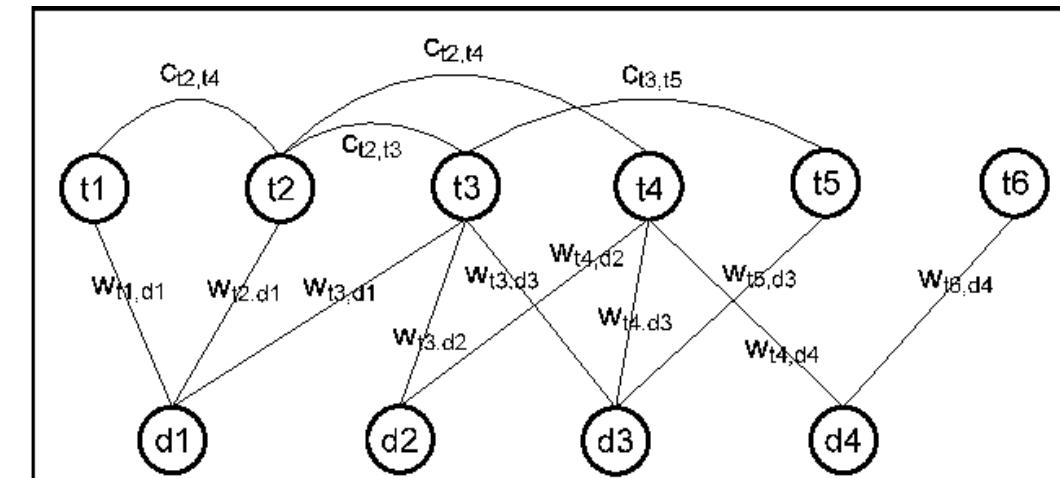
Regular equivalence doesn't care about which connections but to **which set/group a node is connected**

- Two nodes  $u$  and  $v$  are regularly equivalent if they are **equally related to equivalent others**
- Assumes a similarity between sets of nodes



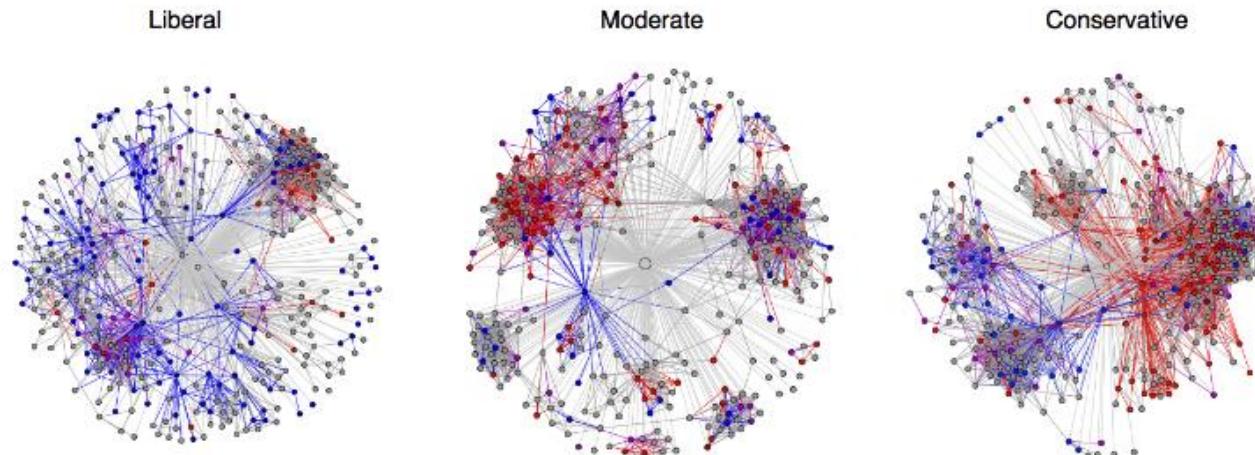
→ *Interact*

→ *Collaborative-Based Filtering Recommendation*

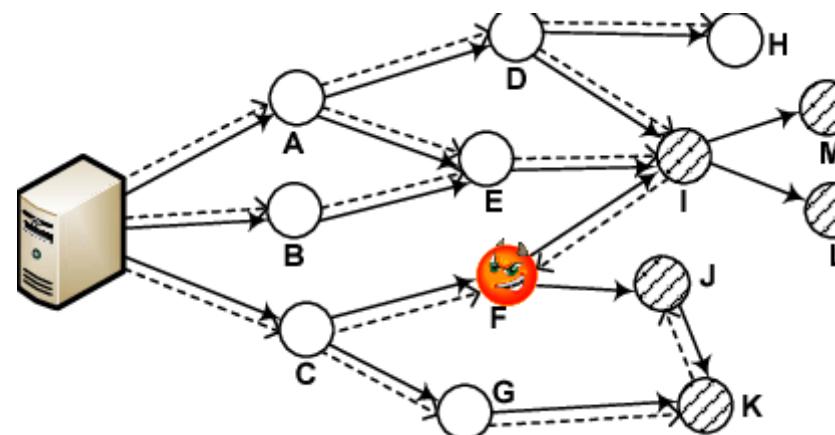


## ➤ Social Network Analysis

- Identifying user roles or community affiliations or politics group in social networks

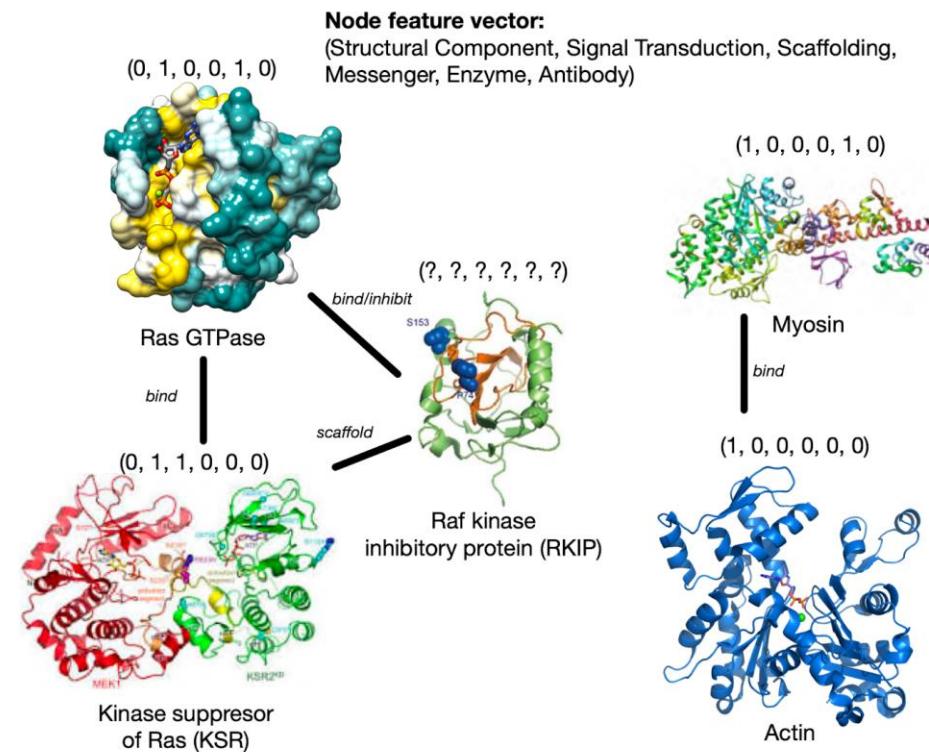


- Detecting fake accounts or malicious users in online social networks

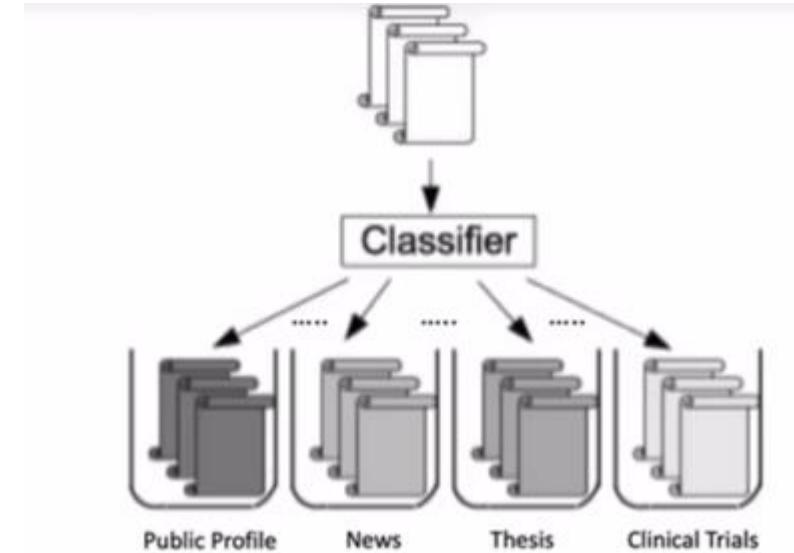
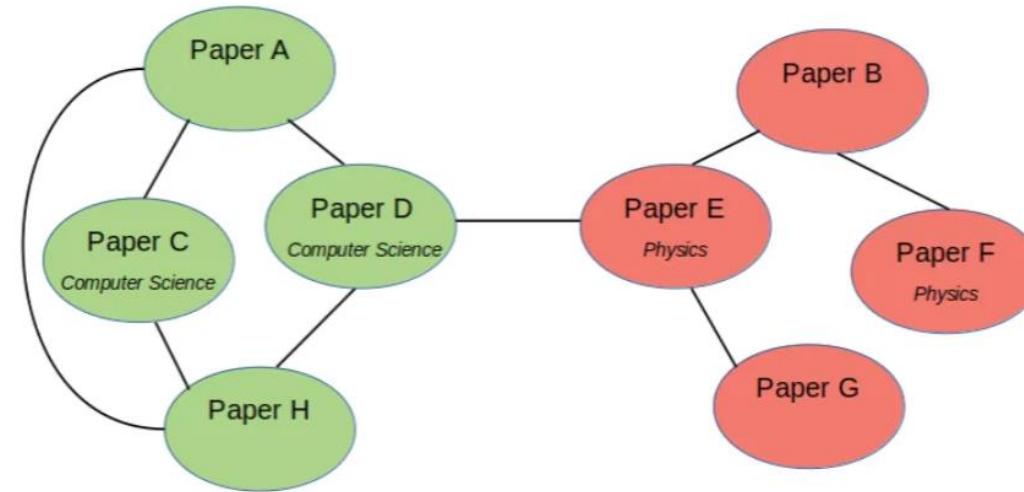


## ➤ Biological Networks:

- Classify protein functions
- Identifying biomarkers or disease subtypes in biomedical networks for personalized medicine

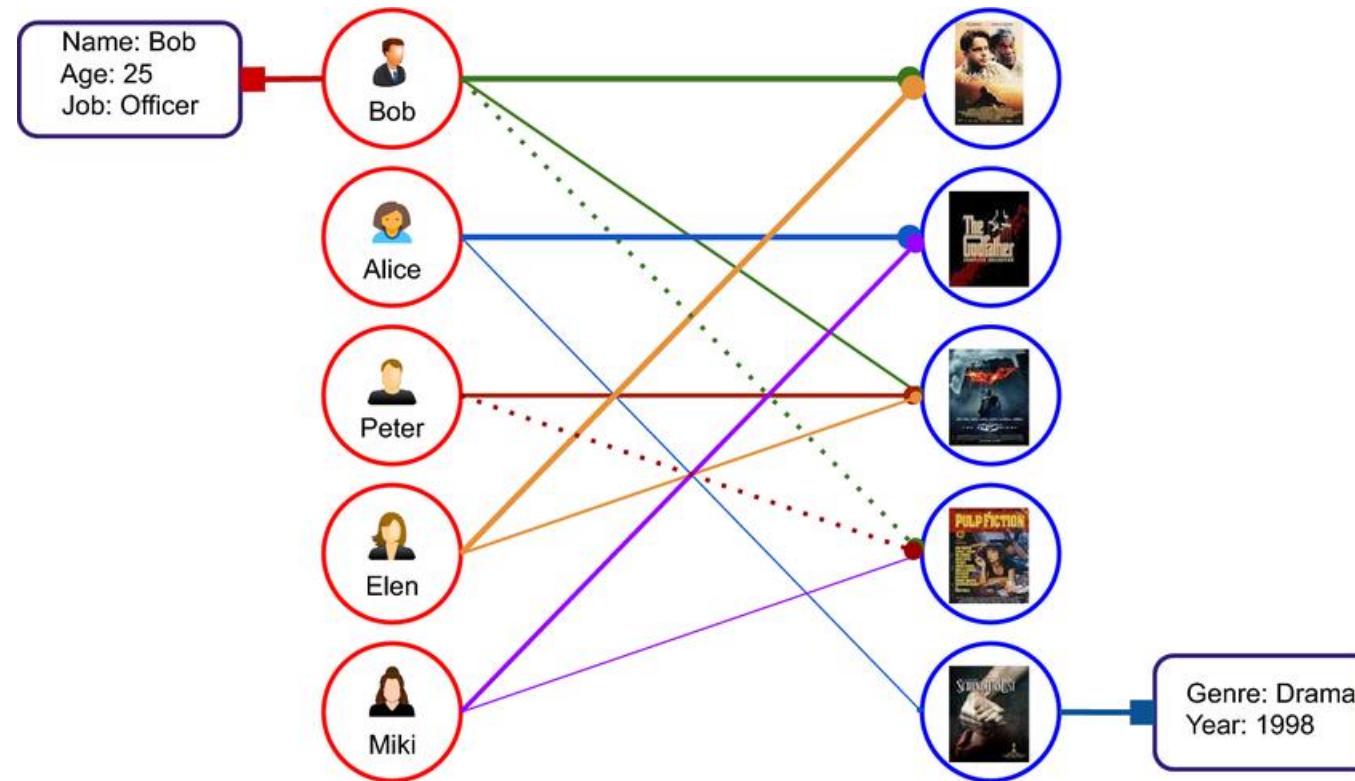


- Web and Information Network:
  - Classifying web pages or documents into categories or topics



## ➤ Recommendation Systems:

- Predicting user preferences or ratings for items (e.g., movies, products)



- Precision/Recall
- Accuracy + Weighted Loss
- ROC and AUC

- When evaluating a search tool or a classifier, we are interested in at least two performance measures:
- **Precision:** Within a given set of positively-labeled results, the fraction that were true positives =  $TP/(TP + FP)$
- **Recall:** Given a set of positively-labeled results, the fraction of all positives that were retrieved =  $TP/(TP + FN)$
- Positively-labeled means judged “relevant” by the search engine or labeled as in the class by a classifier
  - TP = true positive, FP = false positive
  - FN = false negative, TN = true negative

		ACTUAL VALUES	
		Positive	Negative
PREDICTED VALUES	Positive	TP	FP
	Negative	FN	TN

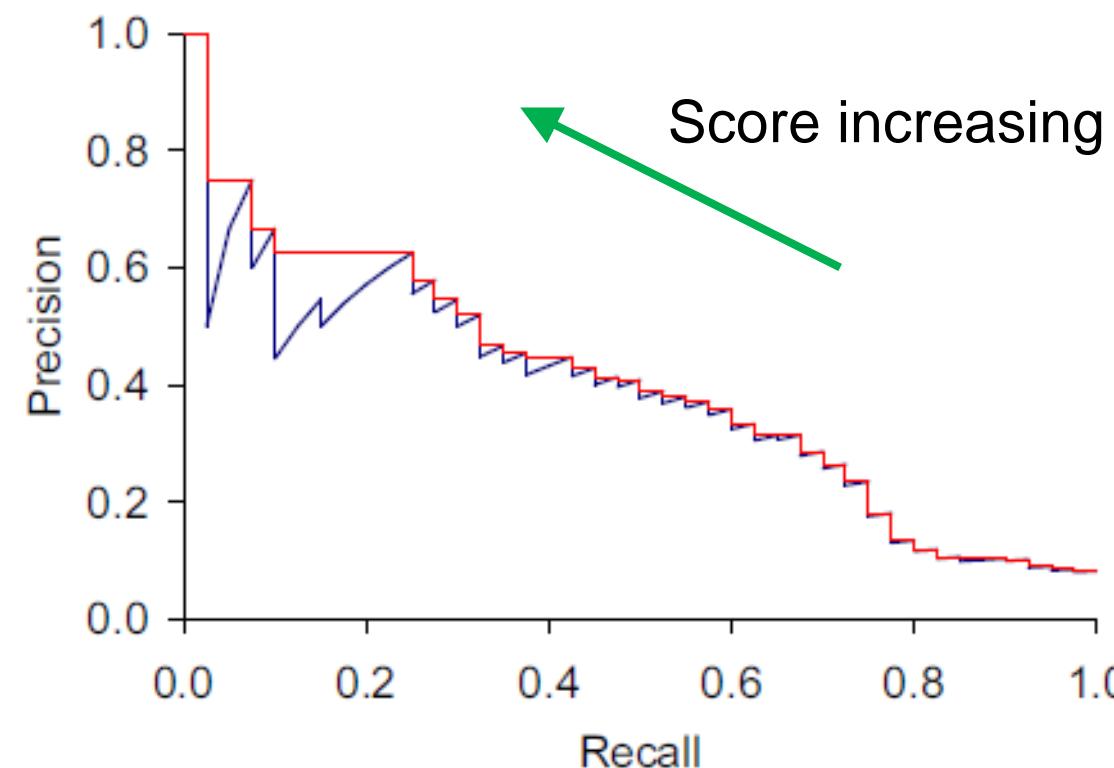
The predicted value is positive and its positive

Type I error : The predicted value is positive but it False

Type II error : The predicted value is negative but its positive

The predicted value is Negative and its Negative

- Search tools and classifiers normally assign **scores** to items
- Sorting by score gives us a precision-recall plot which shows what performance would be for **different score thresholds**



- The simplest measure of performance would be the fraction of items that are correctly classified, or the “accuracy” which is:

$$\frac{TP + TN}{TP + TN + FP + FN}$$

- But this measure is **dominated by the larger set (of positives or negatives)** and favours trivial classifiers
- e.g. if 5% of items are truly positive, then a classifier that always says “negative” is 95% accurate

- We can instead try to minimize a **weight sum**:

$$w_1 \text{ FN} + w_2 \text{ FP}$$

- And typically,  $w_1 \gg w_2$ , since positives are often much rarer (clicks or purchases or viewing a movie)

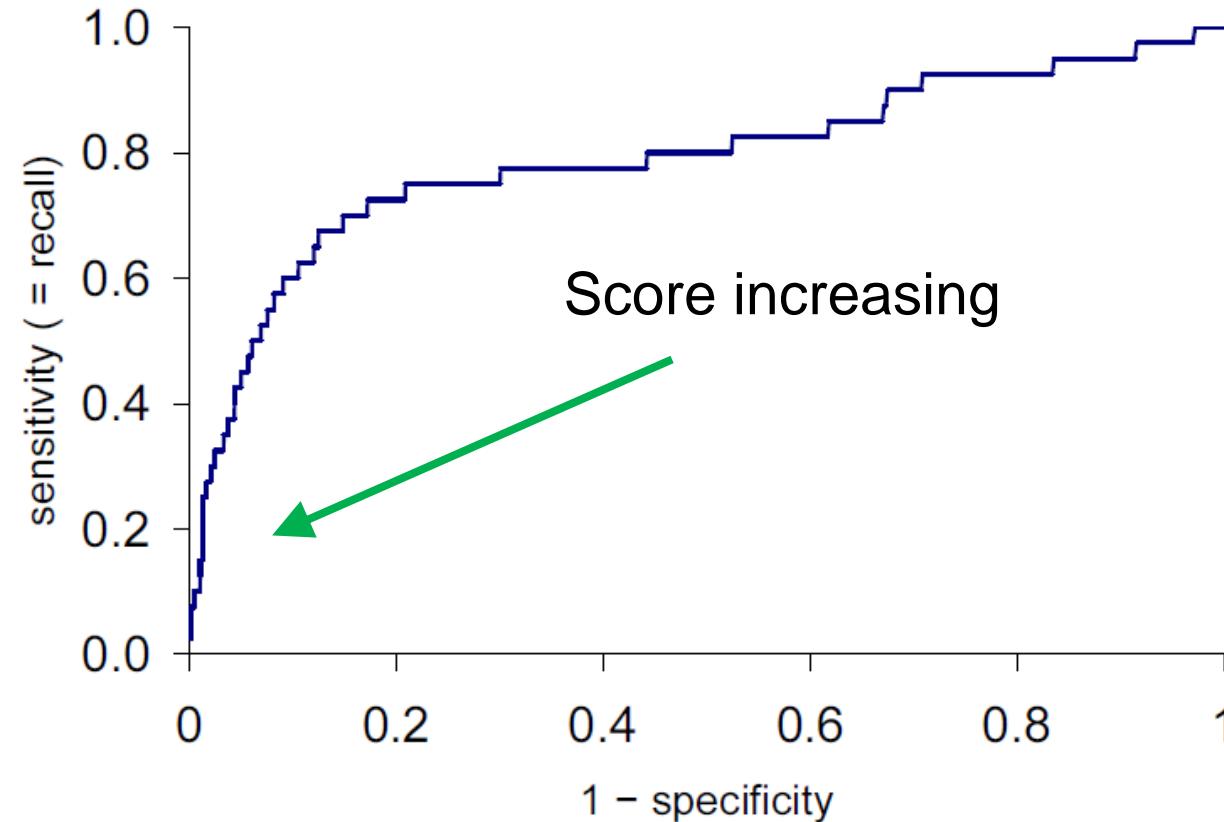
- A measure that naturally combines precision and recall is the  $\beta$ -weighted F-measure:

$$F = \frac{(\beta^2 + 1)PR}{\beta^2P + R}$$

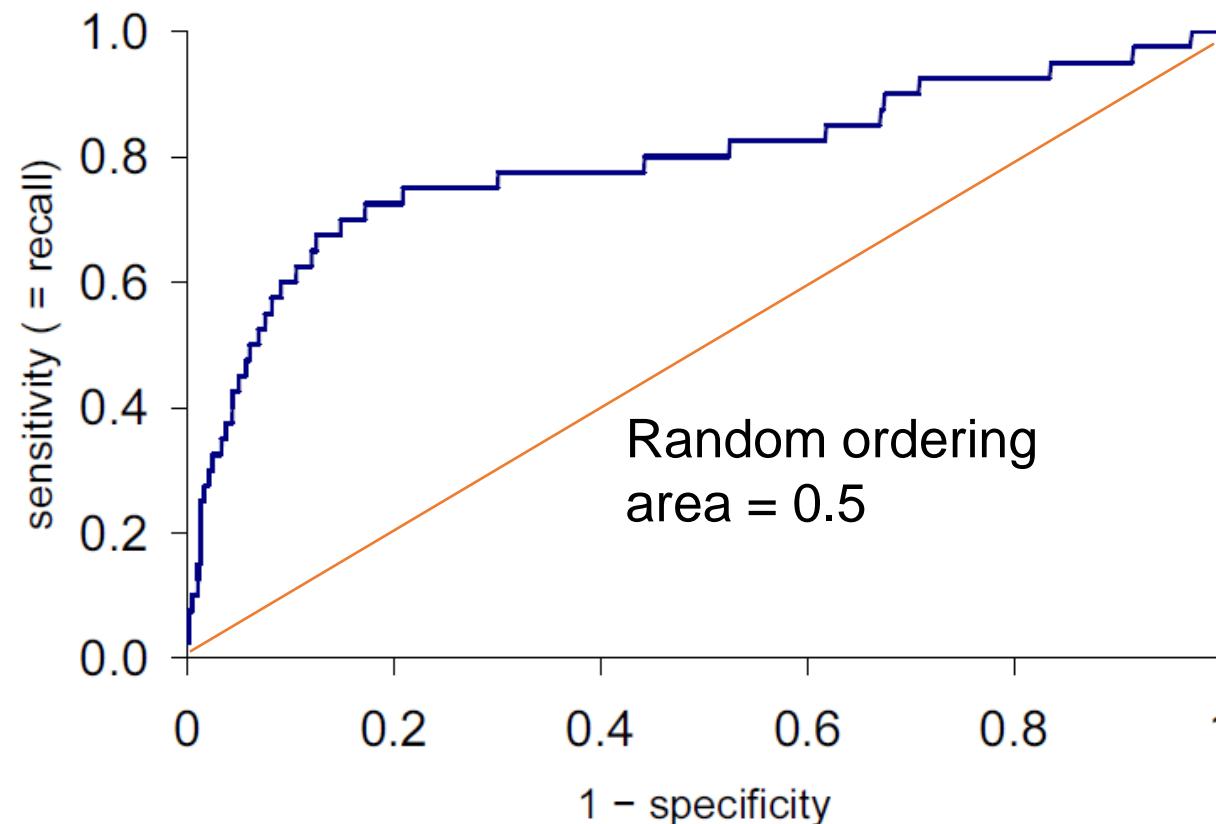
Which is the weighted harmonic mean of precision and recall. Setting  $\beta = 1$  gives us the  $F_1$  – measure. It can also be computed as:

$$F_{\beta=1} = \frac{2PR}{P + R}$$

- ROC is Receiver-Operating Characteristic
- ROC plots:
  - Y-axis: true positive rate =  $TP/(TP + FN)$ , same as recall
  - X-axis: false positive rate =  $FP/(FP + TN) = 1 - \text{specificity}$



- ROC AUC is the “Area Under the Curve” – a single number that captures the overall quality of the classifier
- It should be between 0.5 (random classifier) and 1.0 (perfect)





네트워크 과학 연구실  
NETWORK SCIENCE LAB



가톨릭대학교  
THE CATHOLIC UNIVERSITY OF KOREA

