Feature Engineering and Node Classification

Prof. O-Joun Lee

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

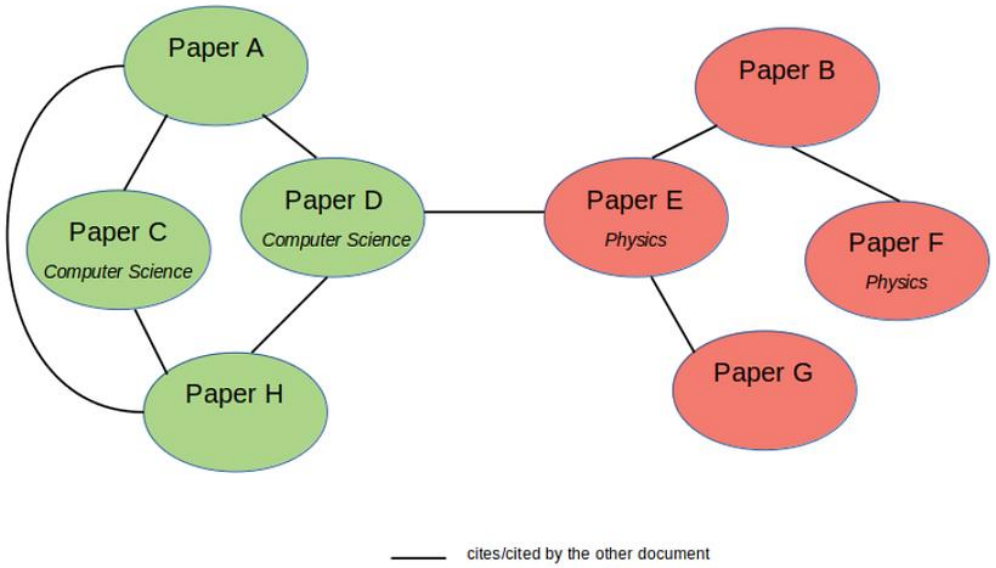
네트워크 과학 연구실
NETWORK SCIENCE LAB

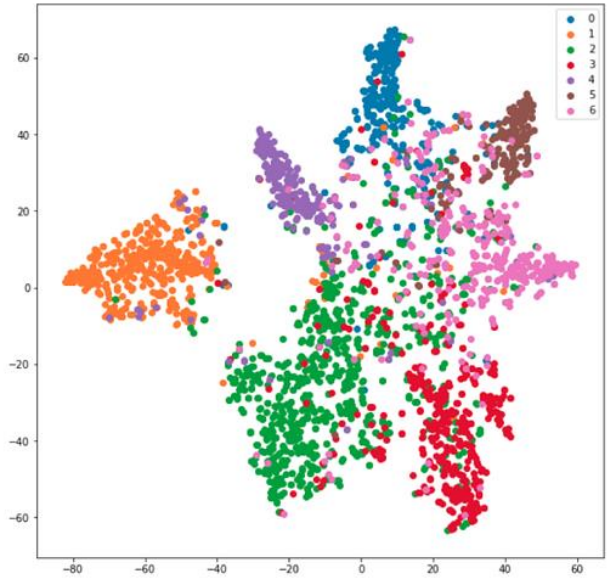
가톨릭대학교
THE CATHOLIC UNIVERSITY OF KOREA

Contents

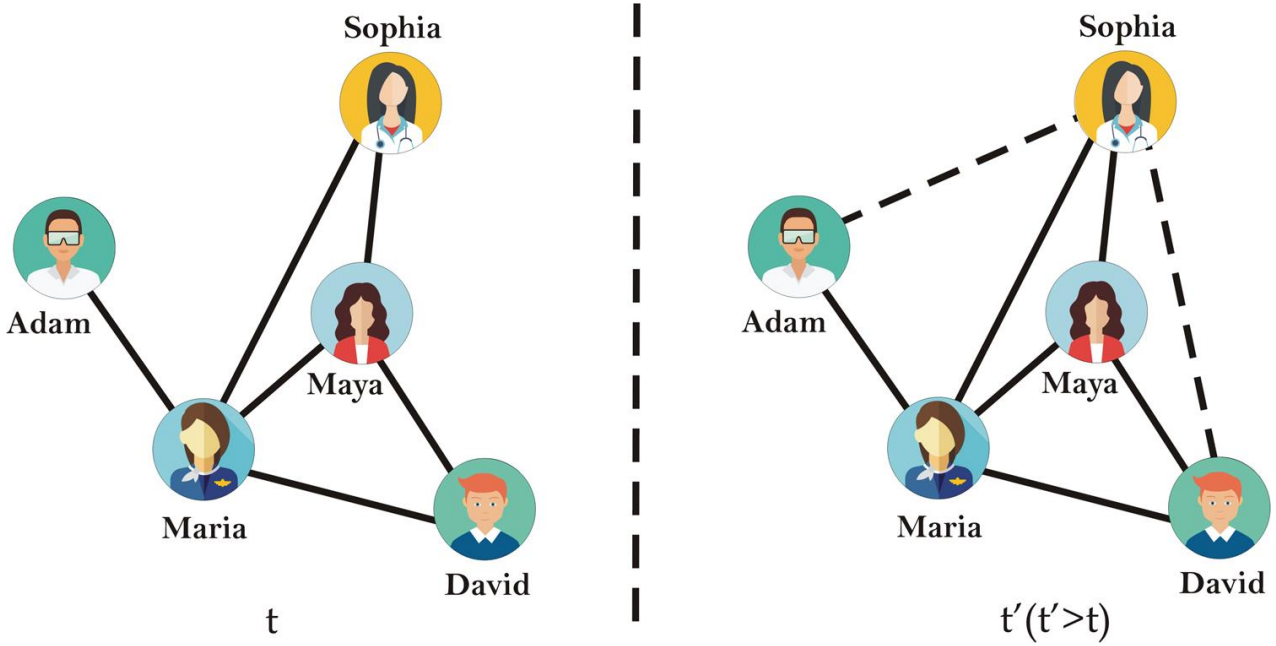
- Overview of Machine Learning pipeline.
- Feature Extraction Methods: How to extract structures for each node?
  - Node-level Features.
    - Local features: Node degree, centrality, clustering coefficient, neigbhoring information.
    - Structure-based features: Graphlet, similarity, label propagation, embeddings.
  - Edge-level features:
    - Distances, Local and Global features.
- Node classification Task.
- Evaluation Metrics:

- One of main machine learning task: classification.
  - Node-level classification:
    - Predicting the classes or labels of nodes.
    - For example, detecting the paper field in a citation network.



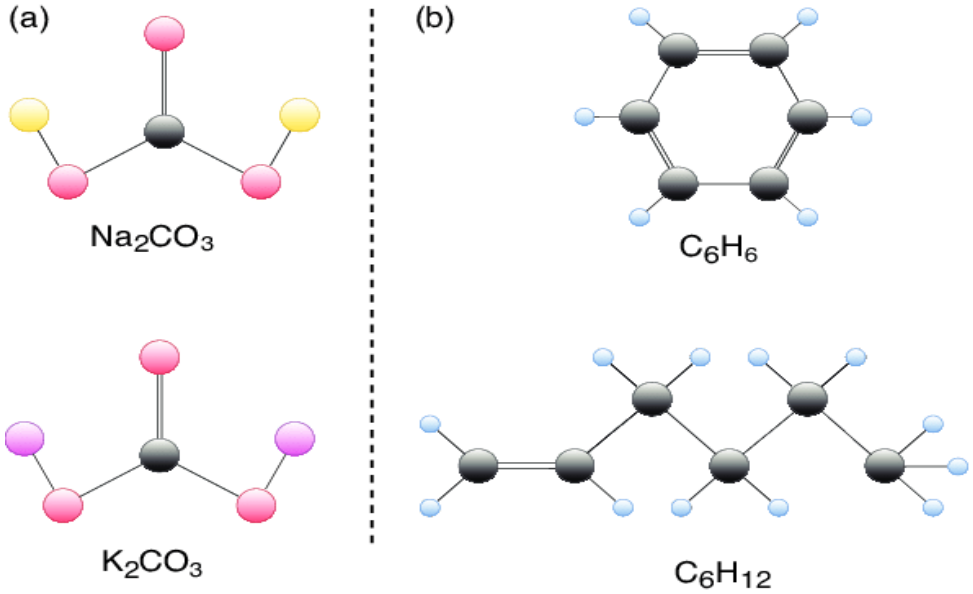


- One of main machine learning task: classification.
  - Edge-level or Link-level classification:
    - Predicting the classes or labels of nodes.
    - For example, a social networking service suggests possible friend connections based on network data.



https://www.nature.com/articles/s41598-019-57304-y

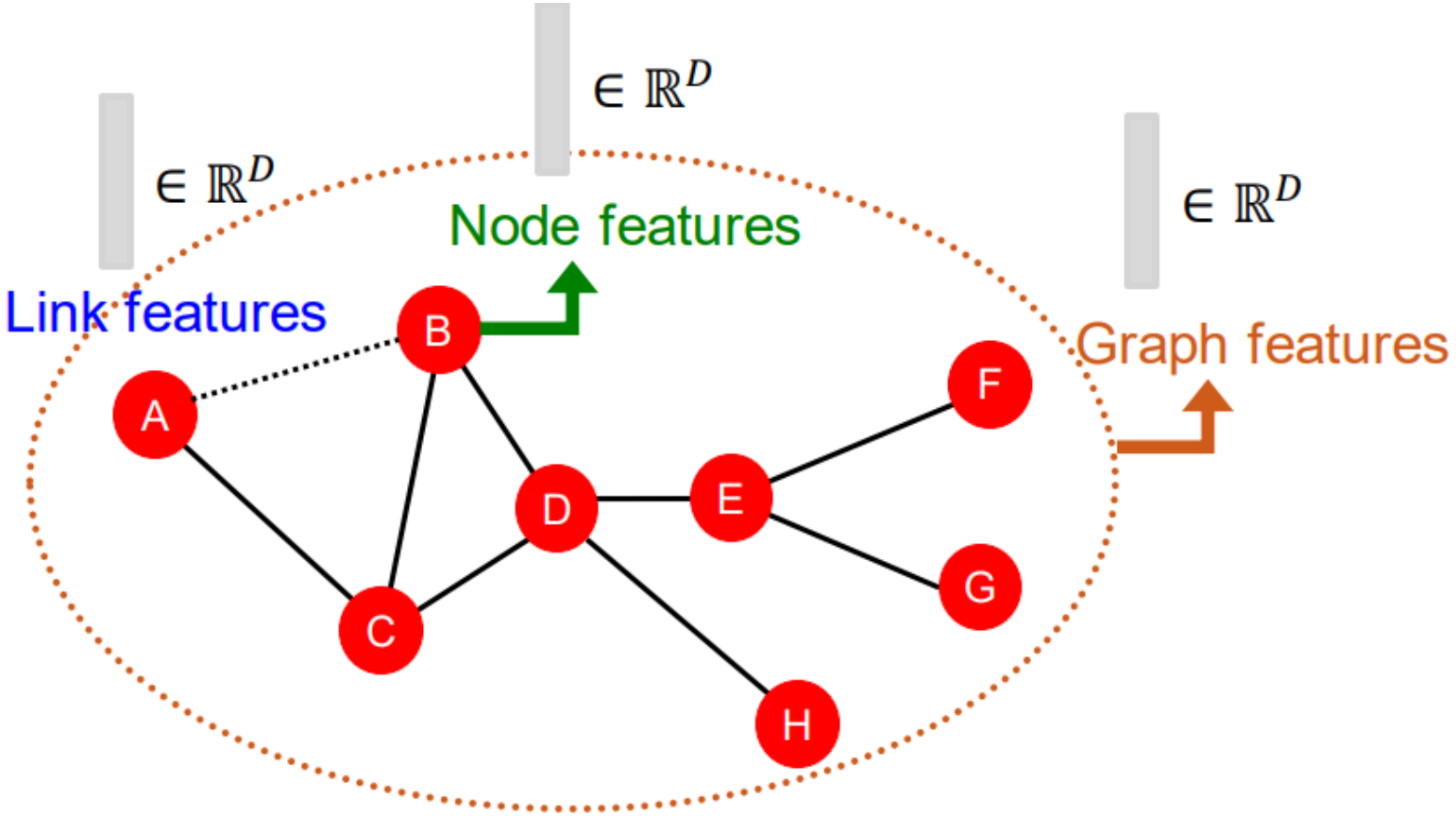
- One of main machine learning task: classification.
  - Graph-level classification:
    - Classifying a graph itself into different categories.
    - Inputs: A collection of graphs.
    - For example, determining if a chemical compound is toxic or non-toxic by looking at its graph structure.



https://www.researchgate.net/figure/An-illustration-of-molecule-graph_fig1_349880436

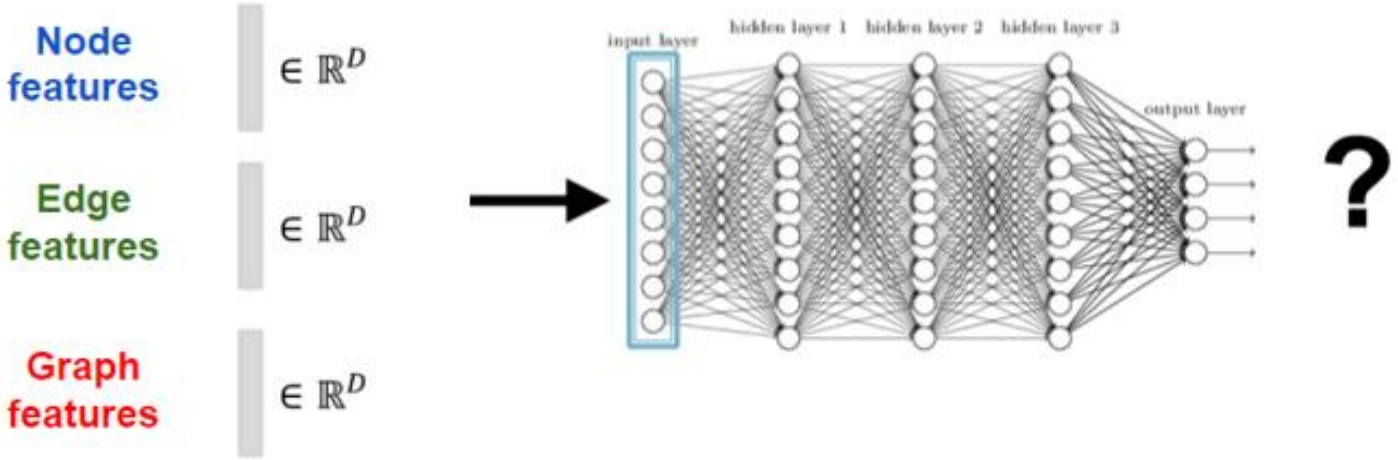
Overview of Machine Learning Pipeline

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



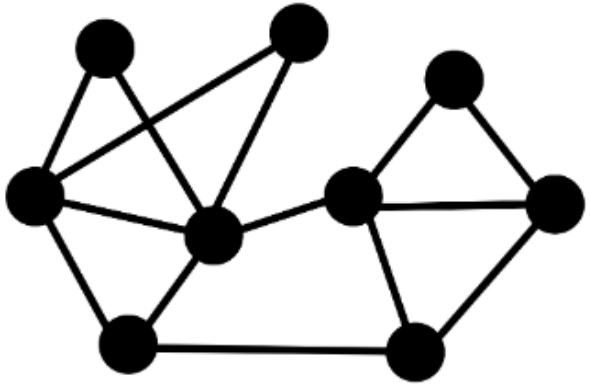
Traditional Machine Learning pipeline

- **Feature extraction**:
  - Node features
  - Edge features
  - Graph features
- **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



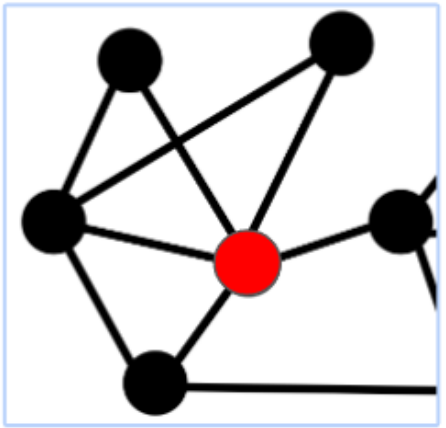
Feature Extraction Design

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



Graph

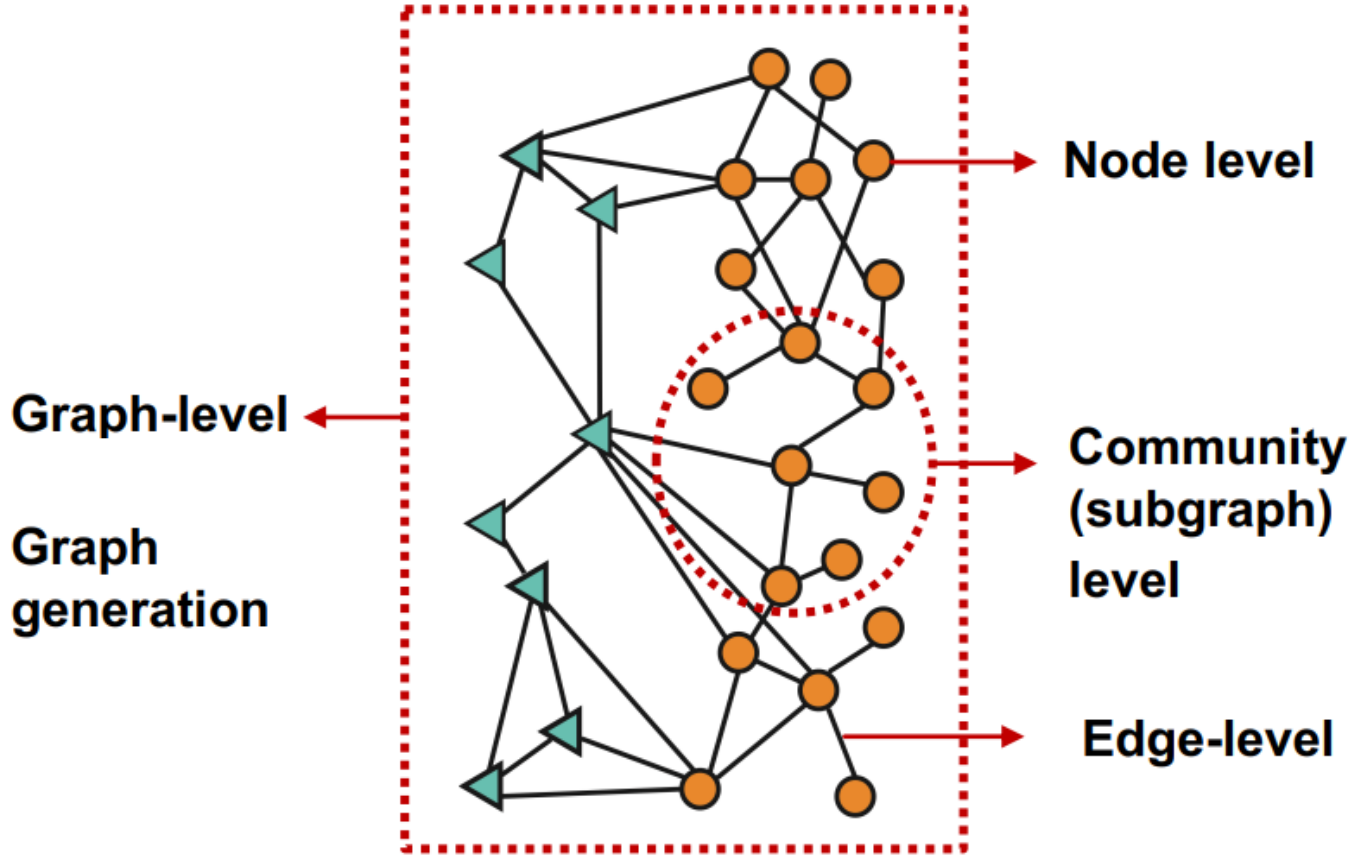
Node prediction



With neighboring topology

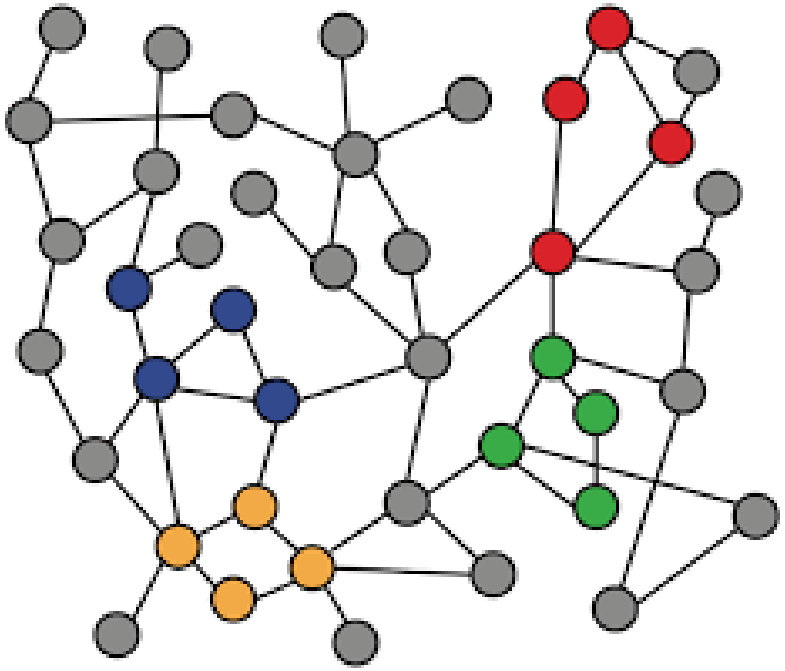
Feature Extraction Approaches

- Three main types of graph components:
  1. Node-level features.
  2. Edge-level features.
  3. Graph-level features (Next week).



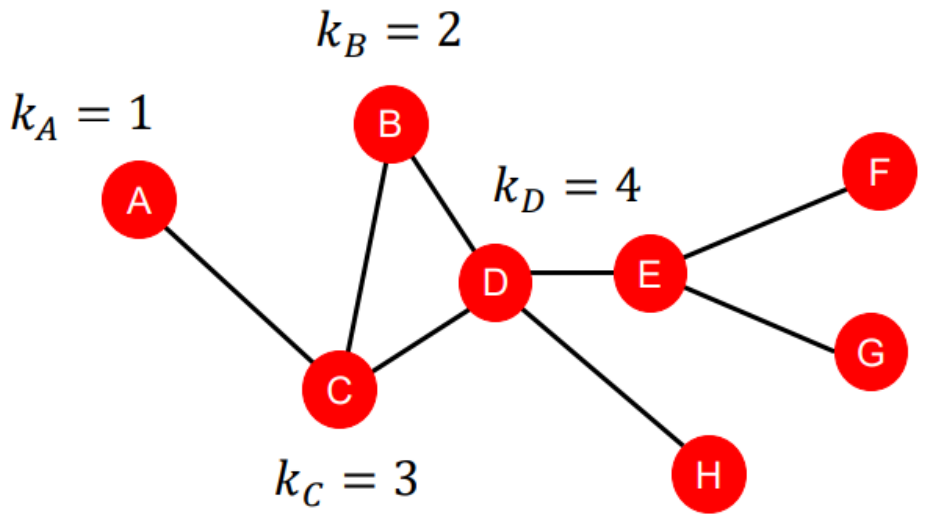
Node-level Features: Overview

- **Goal**: Characterize the structure and position of a node in the network:
  - Local features:
    - Node degree.
    - Node centrality.
    - Vector similarity
    - Clustering coefficient.
    - Neighboring information.
  - Structure-based features:
    - Graphlets.
    - SimRank.
    - Label Propagation.



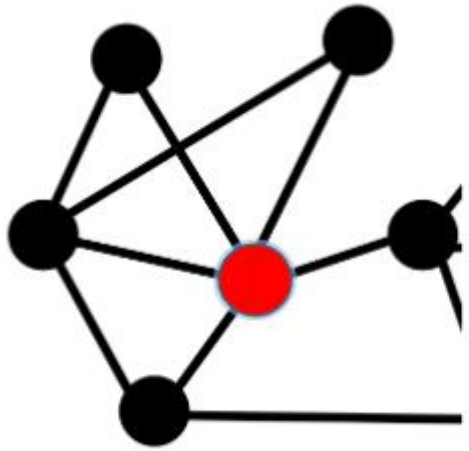
Node Features: Node Degree

- 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 Features: Node Centrality

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



Node Features: Node Centrality (1)

➢ Eigenvector Centrality

- ➢ A node v is important if surrounded by important neighboring nodes u ∈ N(v).
- ➢ We model the centrality of node v as the sum of the centrality of neighboring nodes recursively:

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

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

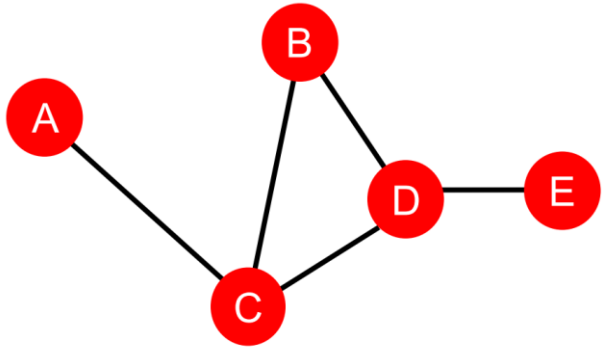
- ➢ We can see that centrality ***c*** is the eigenvector for ***A***

Node Features: Node Centrality (2)

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

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

  - For example:



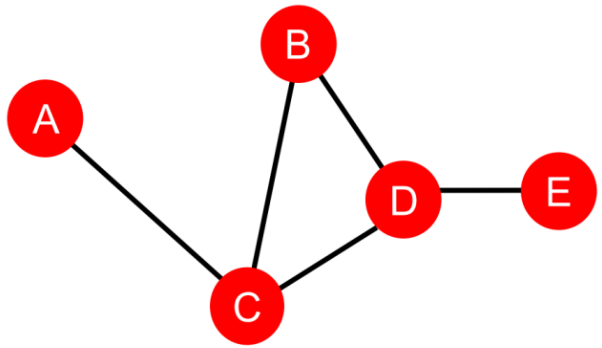
- $C_A = C_B = C_E = 0$
- $C_C = 3$ (A-C-B, A-C-D, A-C-D-E)
- $C_D = 3$ (A-C-D-E, B-D-E, C-D-E)

Node Features: Node Centrality (3)

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

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

  - 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:**
  - computes 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:

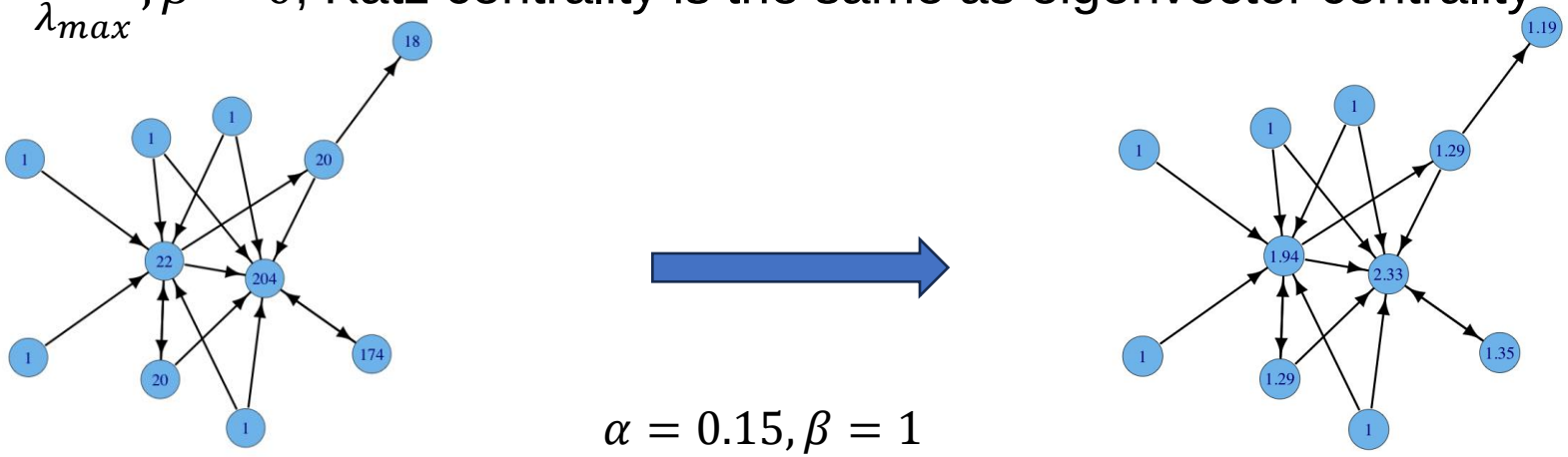
$$x_i = \alpha \sum_j A_{ij} x_j + \beta,$$

where:

α is a constant called damping factor, and β is a bias constant,

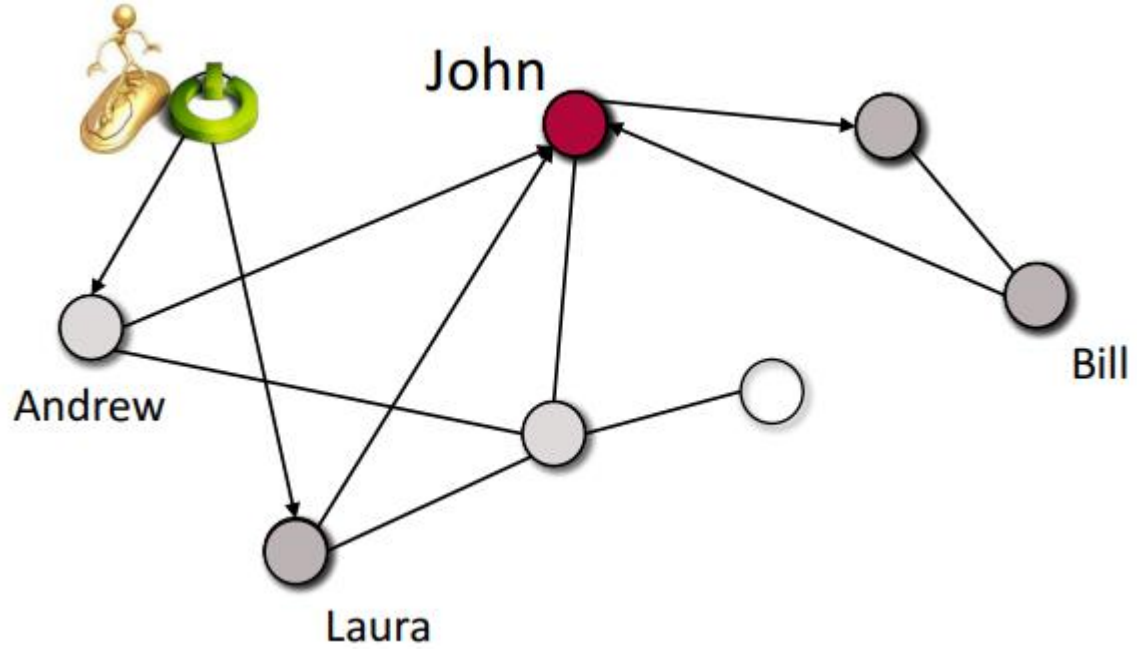
A is the adjacency matrix.

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



https://www.sci.unich.it/~francesc/teaching/network/katz.html

Node Features: PageRank



- The importance of John is high **<u>if</u>** Laura, Andrew, and Bill are also important.

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

Node Features: Vector similarity

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

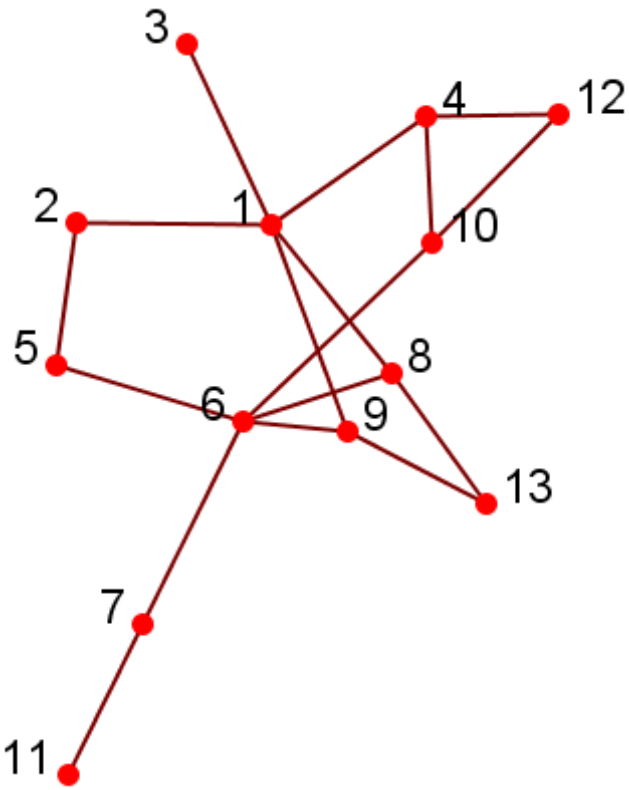
	1	2	3	4	5	6	7	8	9	10	11	12	13
A vector → 5		1				1							
Structurally equivalent: 8	1					1							1
Structurally equivalent: 9	1					1							1

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

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

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

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



Node Features: Clustering Coefficient

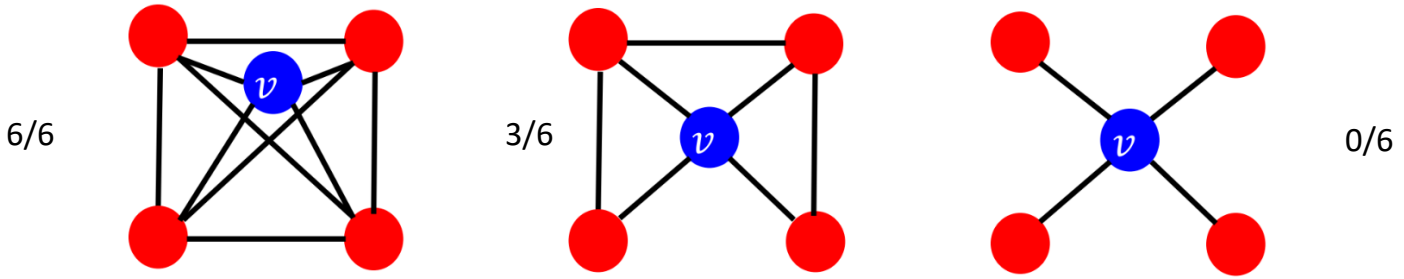
- Measures **how connected v's neighboring nodes** are:

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

#(node pairs among k_v neighboring nodes)

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

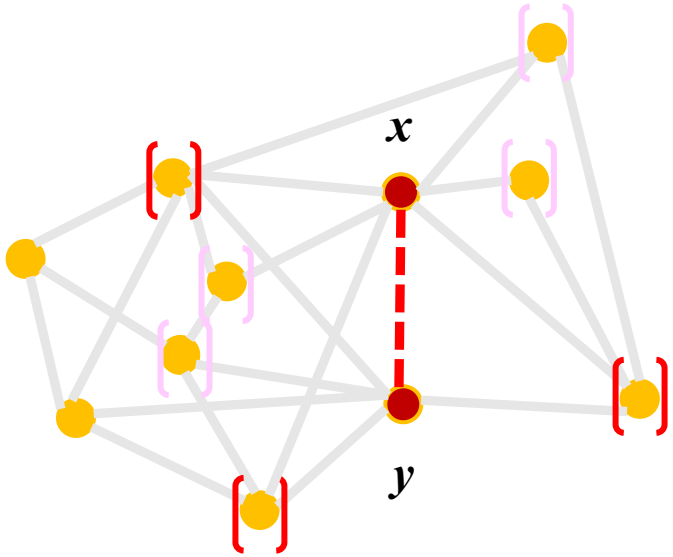
- For example:



$e_v = 1$ $e_v = 0.5$ $e_v = 0$

Node Features: Neighboring information

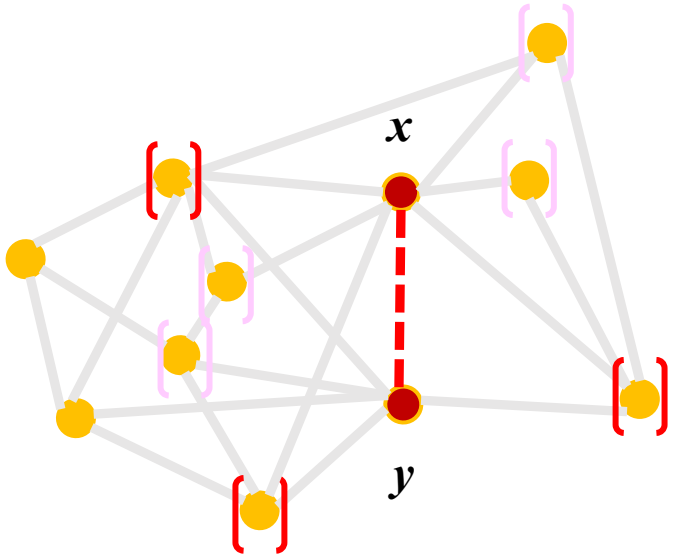
- Common neighbors:
  - How many neighbours are in common between x and y.
  - X and Y have 3 common neighbours, more likely to collaborate.
  - Let $N(x)$ denote the set of nodes adjacent to x, $N(x) = \{m | (x, m) \in E\}$.



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

Node Features: Neighboring information (2)

- Jaccard coefficient:
  - How likely a neighbour of x is also a neighbour of y.
  - Same as common neighbors, adjusted for degree.

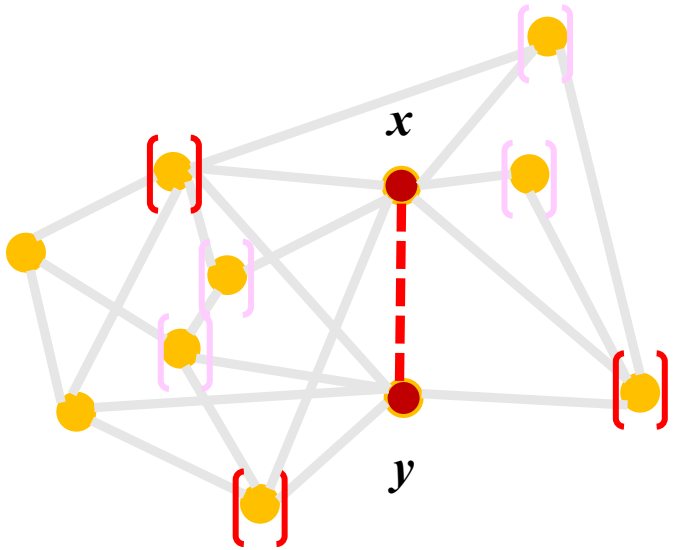


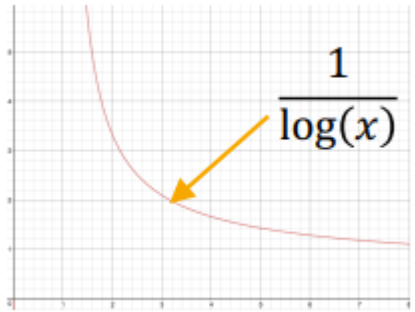
$$JC = \frac{\left|N(x) \cap N(\mathrm{y})\right|}{\left|N(x) \cup N(\mathrm{y})\right|} = \frac{CN}{d_x + d_y - CN}$$

Node Features: Neighboring information (3)

- 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

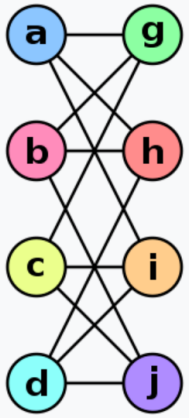
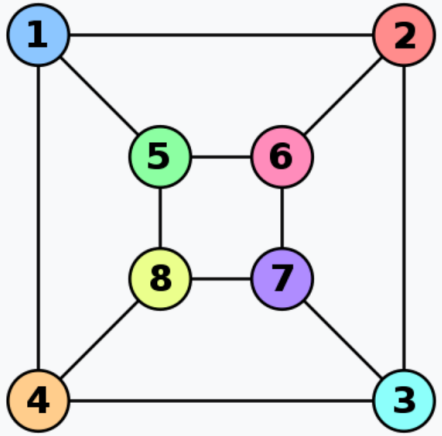
$$AA = \sum_{z \in CN} \frac{1}{\log d_z}$$



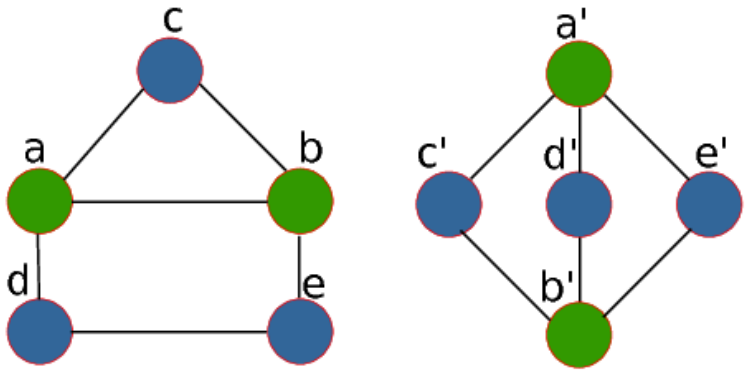


Background: graph Isomorphism

- **Isomorphic graphs** having the same number of vertices, edges, and also 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

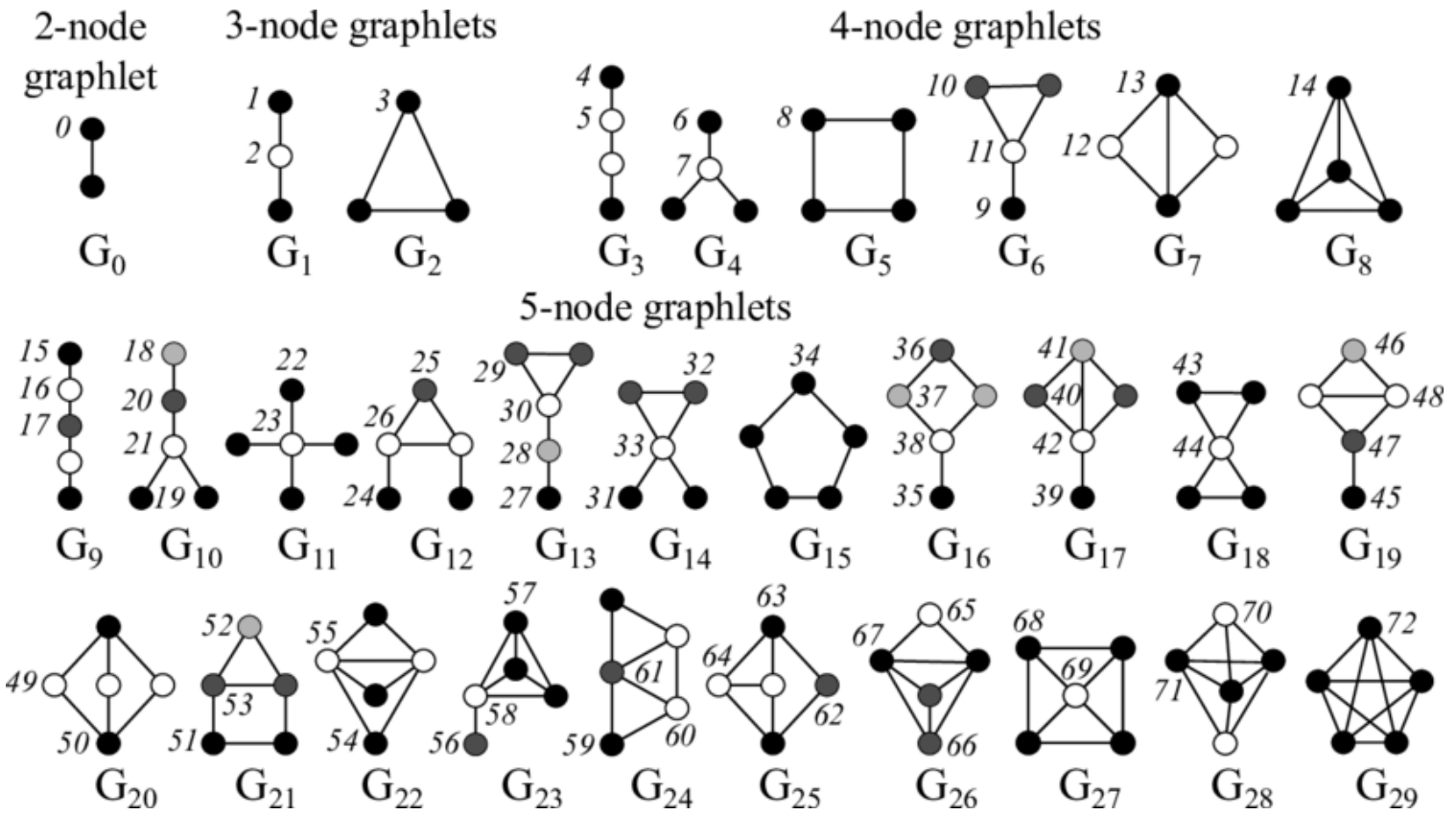


Non-Isomorphic

https://en.wikipedia.org/wiki/Graph_isomorphism

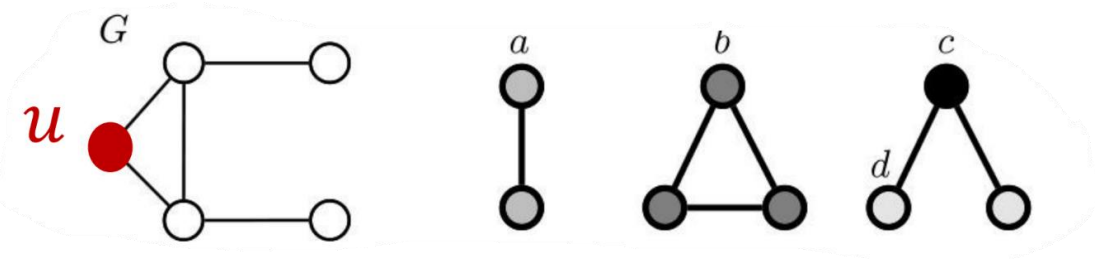
Node Features: Graphlet

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

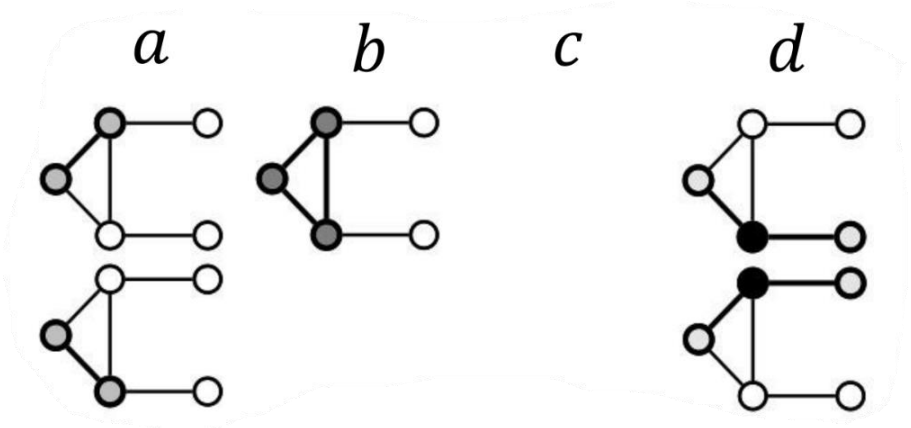


Node Features: Graphlet

- 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).



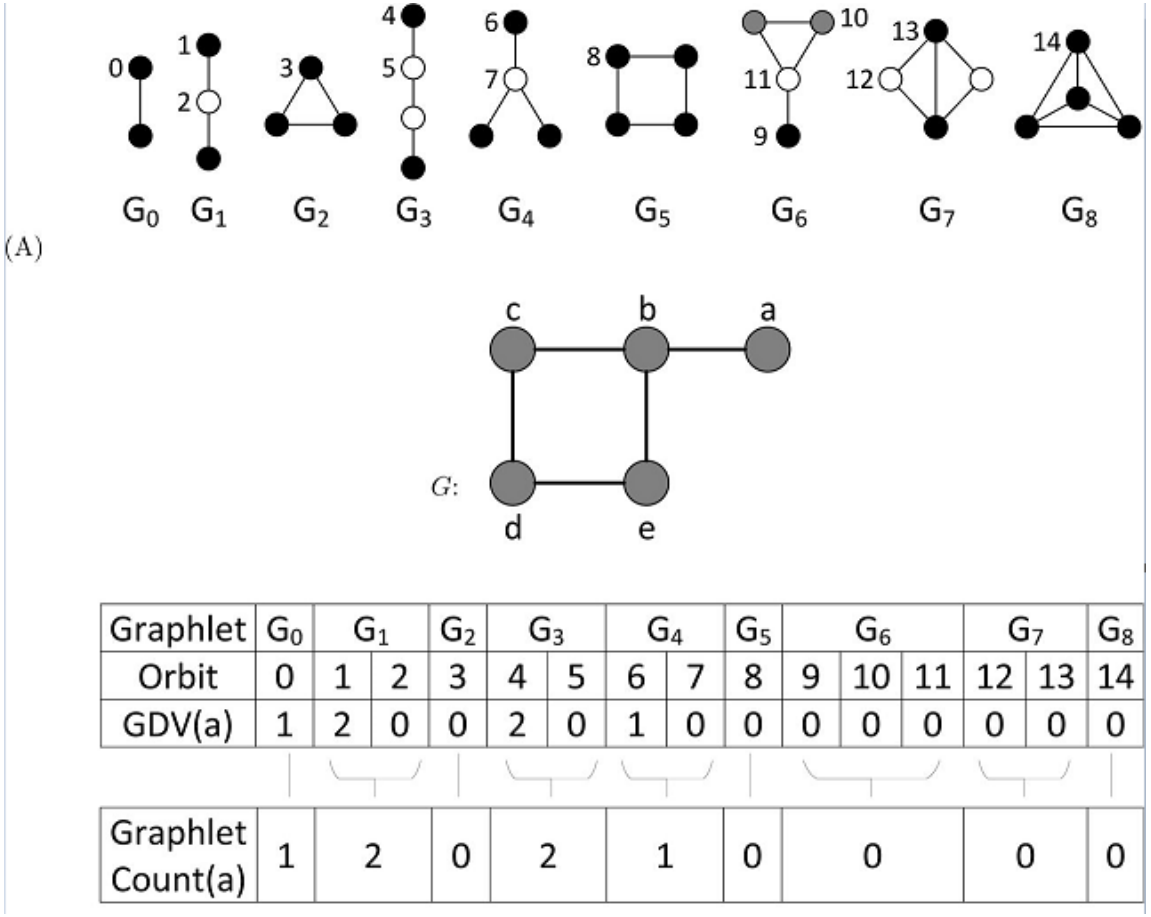
Select graphlets up to 3 nodes



Graphlet instances for node u

Node Features: Graphlet

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



Graphlet	G_0	G_1		G_2	G_3		G_4		G_5	G_6			G_7		G_8
Orbit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
GDV(a)	1	2	0	0	2	0	1	0	0	0	0	0	0	0	0

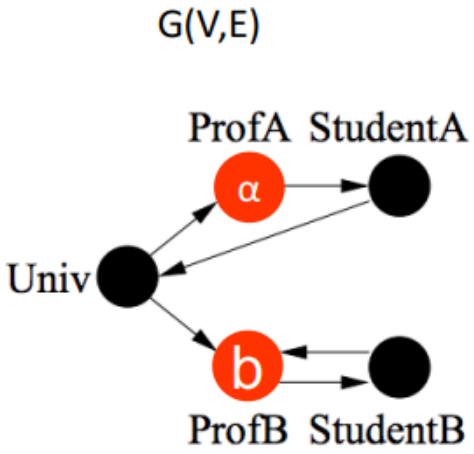
Graphlet Count(a)	1	2	0	2	1	0	0	0	0

Node Features: SimRank

- Idea:
  - Two objects are similar if they are referenced by similar objects.

$$s(a,b) = \frac{\text{decay factor in [0,1]}}{\text{total \# of in-neighbors pairs}} \times \text{Avg similarity between in-neighbors of α and in-neighbors of b}$$

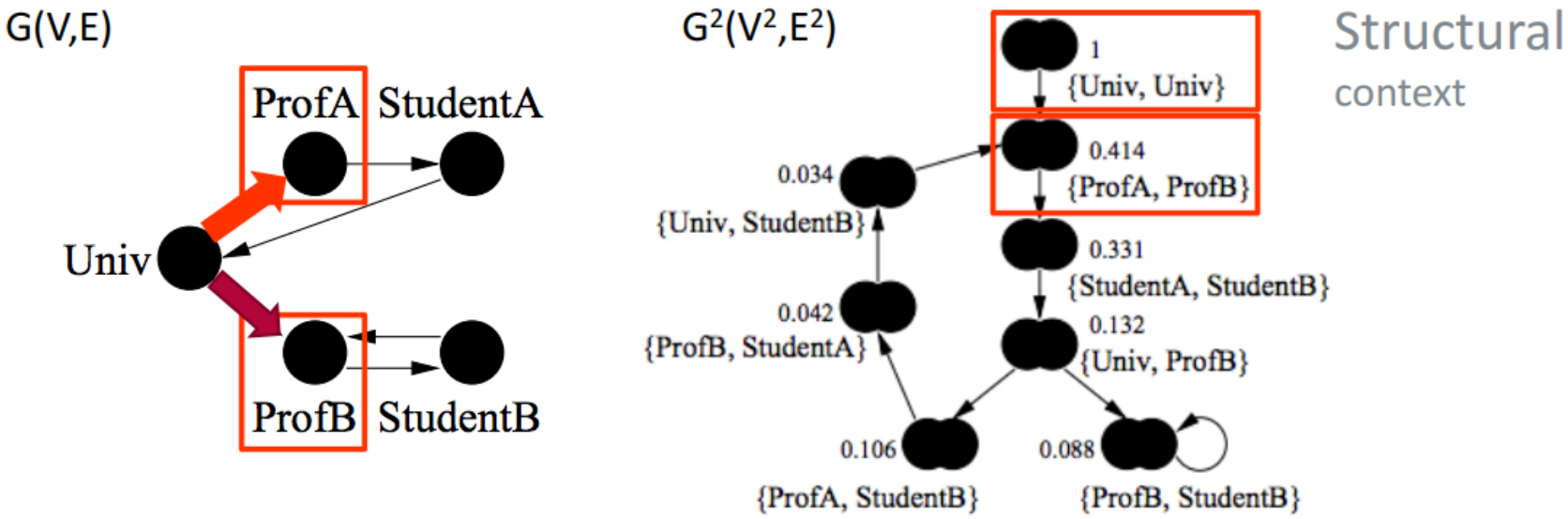
similarity of in-neighbors



Glen Jeh and Jennifer Widom. SimRank: A Measure of Structural-Context Similarity.

SimRank: Intuition

- 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 sources.



Glen Jeh and Jennifer Widom. SimRank: A Measure of Structural-Context Similarity.

Node Features: Learning with Local and Global Consistency

- 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 $(l+1 \le u \le n)$ are unlabelled.
- Output:
  - Predict the label of the unlabelled points

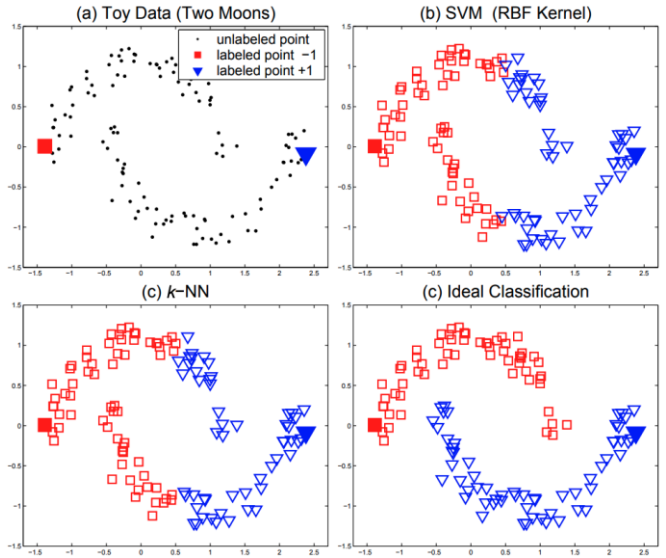
- Let F denote the set of $n \times c$ matrices with nonnegative entries. A matrix $F = [F_1, F_2, \dots, F_n]$ corresponds to a classification on the set X by labelling each point x_i as a label $y_i = argmax_{\{j \le c\}} F_{ij}$.
- We can understand F as a vectorial function $F: X \to R^c$ which assigns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

Node Features: Learning with Local and Global Consistency

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.

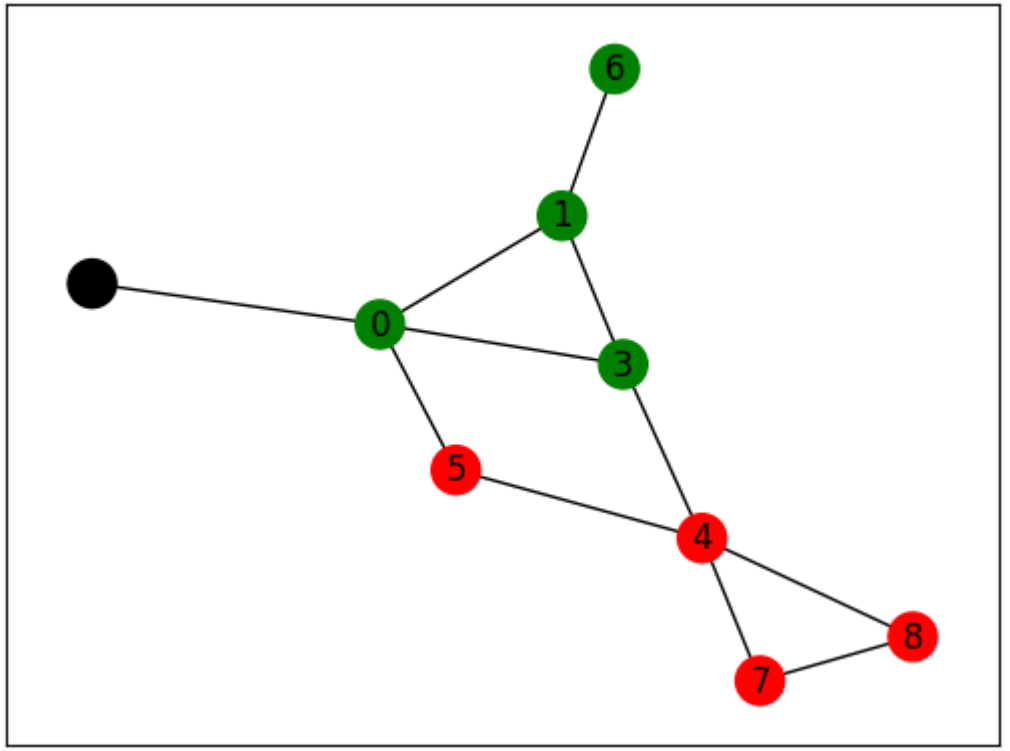
3. Iterate $F(t+1) = \alpha SF(t) + (1-\alpha)Y$ until convergence, where $\alpha \in (0,1)$

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.

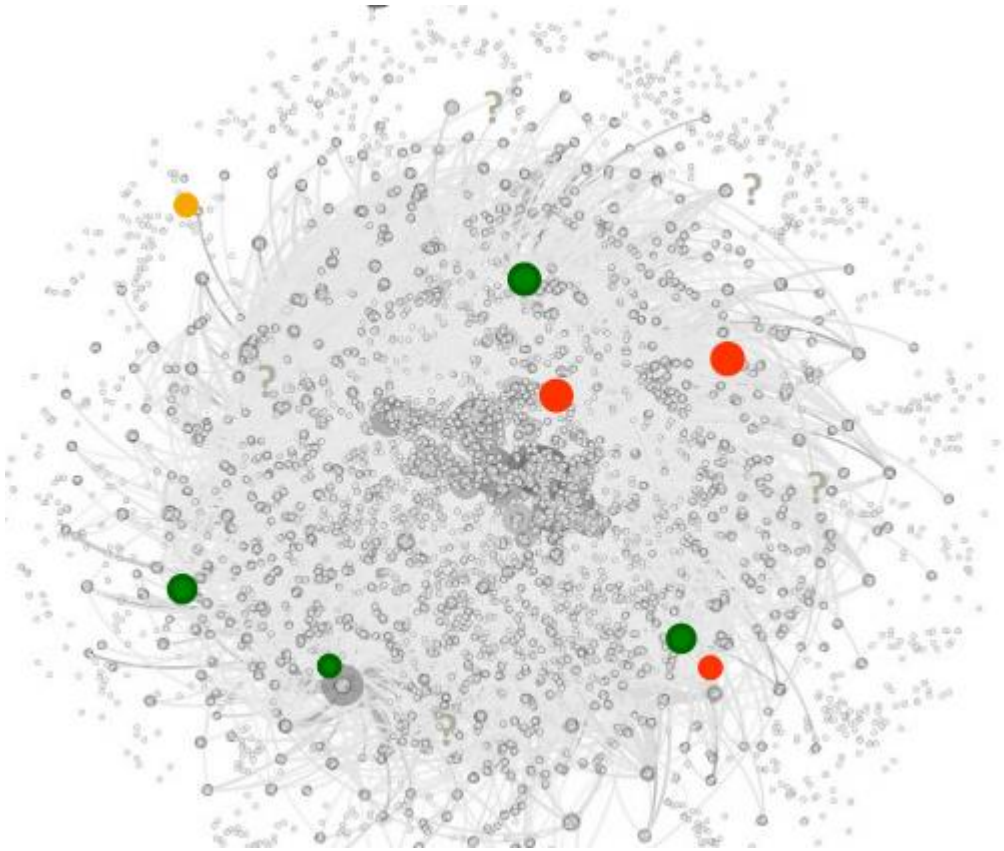
Learning with Local and Global Consistency with SVM: Sample code


```
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
```

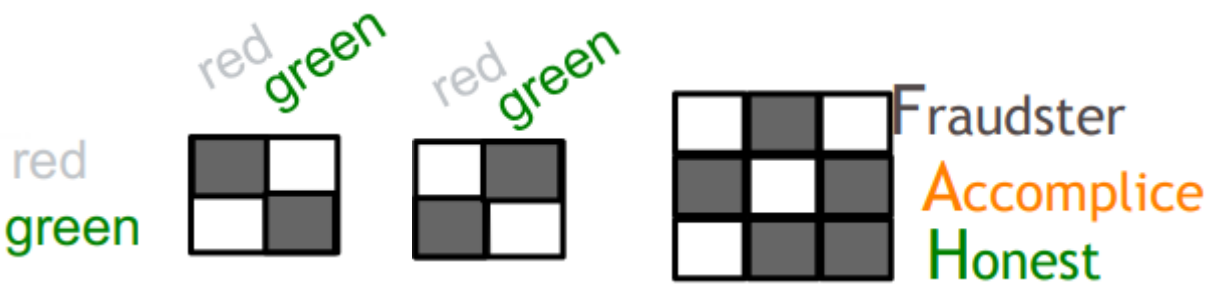
```
['A', 'A', 'A', 'A', 'B', 'A', 'B', 'B', 'B']
```



Background: Guilt-by-Association Techniques



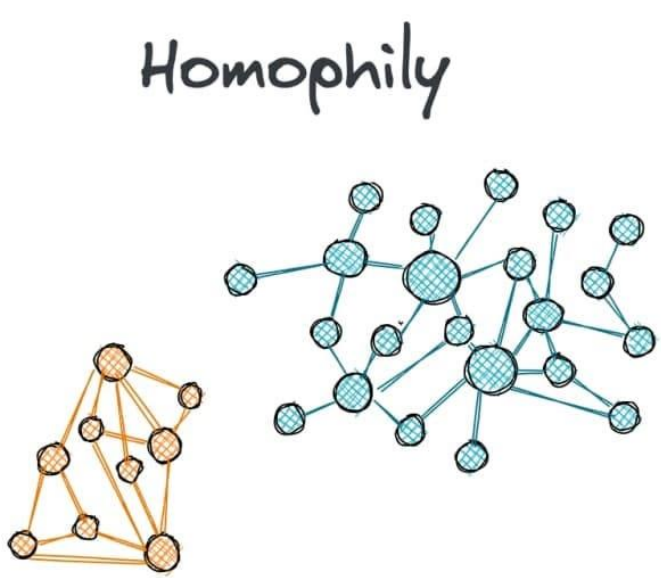
- **Given**:
  - Graph and few labelled nodes.
- **Find**:
  - class (red/green) for rest nodes.
- **Assuming:**
  - Network effect (homophily/heterophily)

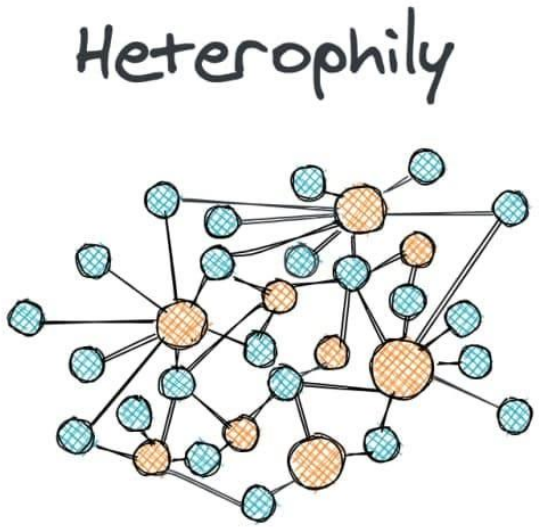


Davide Mottin. Hasso Plattner Institute.

Background: Homophily and Heterophily

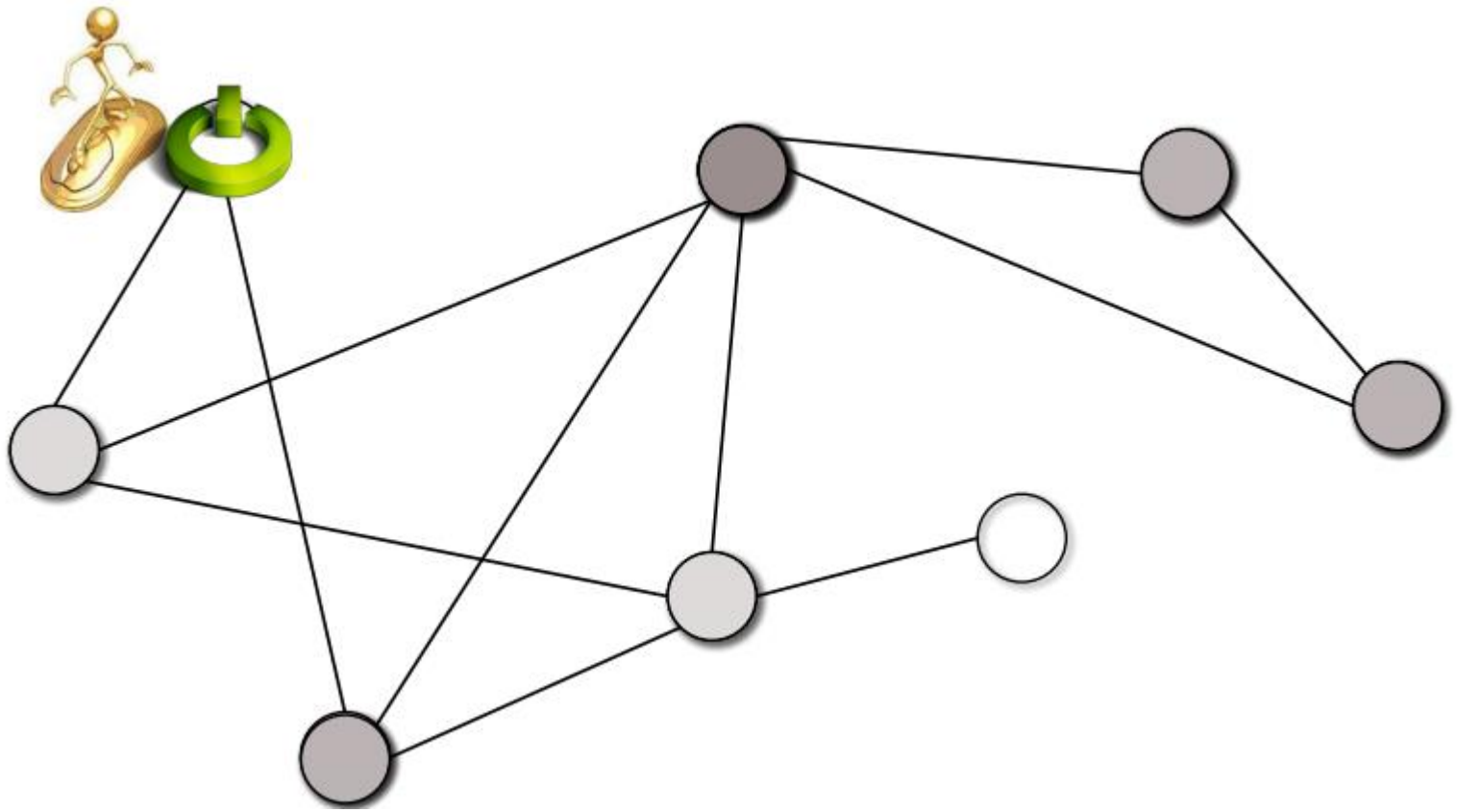
- **Homophily**: the tendency of individuals to associate and bond with similar others.
- **Heterophily**: the tendency to interact with others of different type, while the connectivity within the groups is sparse.





Node Feature: Label Propagation Overview

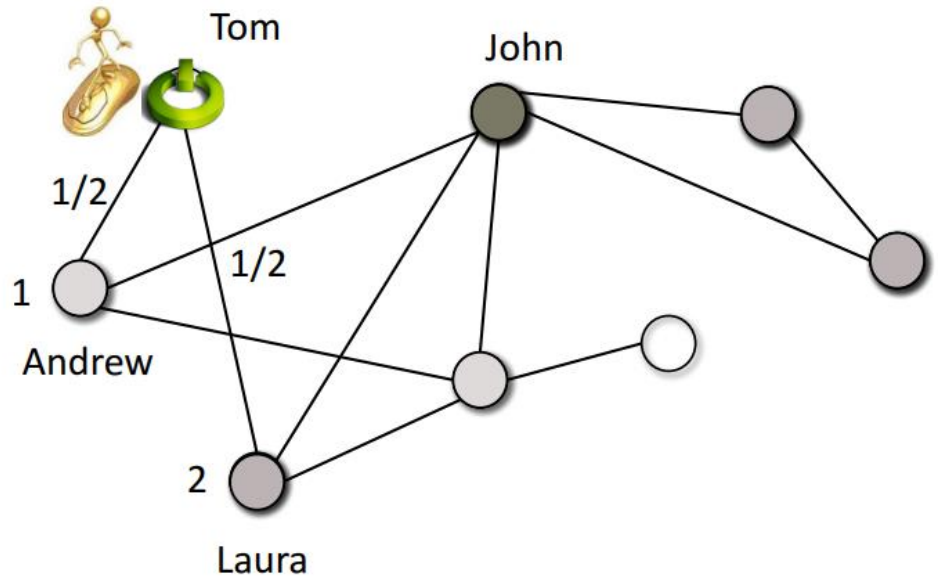
- **Idea**: Propagate labels from a set of nodes to the rest of the graph.



Davide Mottin. Hasso Plattner Institute.

Node Feature: Random Walk

- 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

Node Feature: Personalized RWR

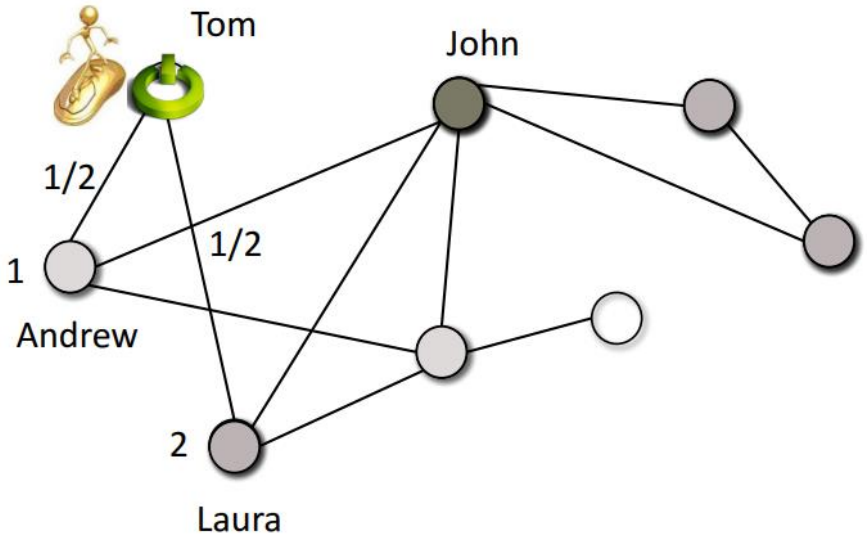
- Now assume that with probability c you perform another move and with probability (1-c) you jump back to Tom.
- Therefore, the probability for the walker of being in Tom place will be:

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

Probability of visiting Andrew at time (t-1)

Number of Andrew and Laura's neighbors

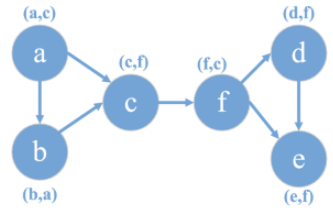
Probability of jumping back to Tom



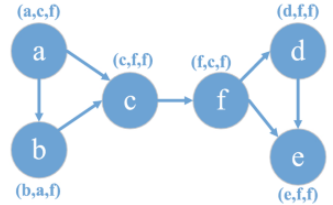
Node Feature: Labels of neighbours of target nodes

- 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 vertex v ∈ V, v updates its label according to:

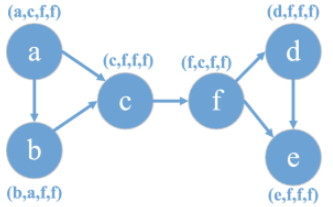
$$l_v = \arg\max \sum_{u \in N(v)} [l_u == l]$$



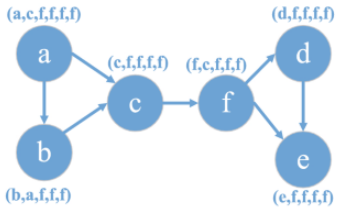
(a) Result after the first label propagation



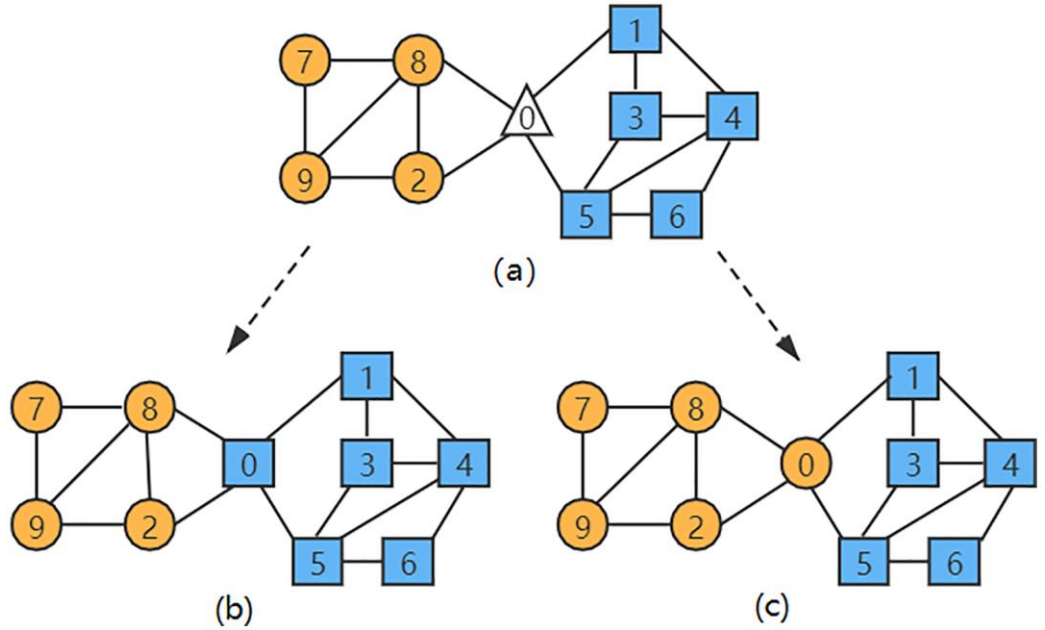
(b) Result after the label second propagation



(c) Result after the third label propagation



(d) Result after the fourth label propagation

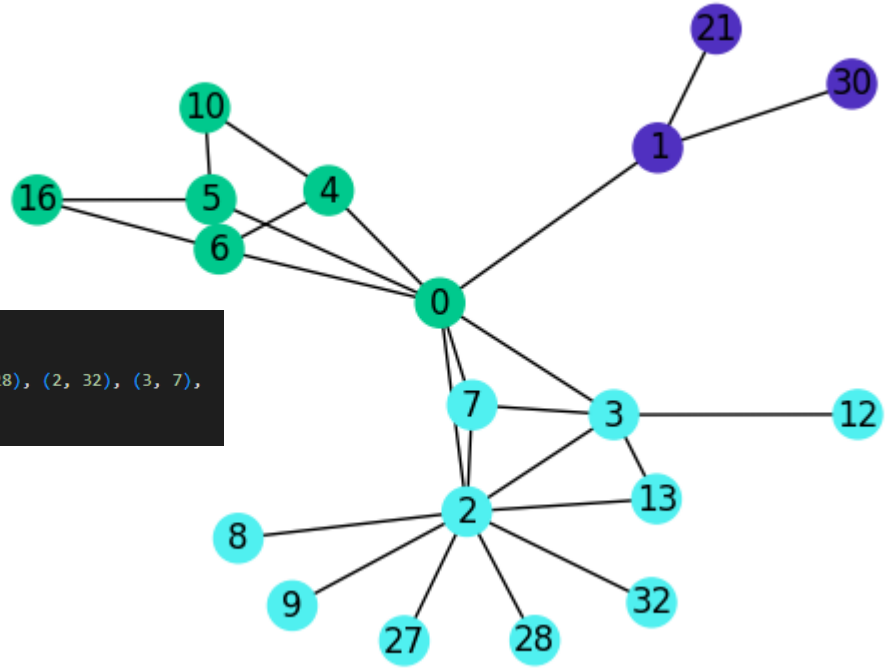


Label propagation: Sample code


```
# 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)])
```

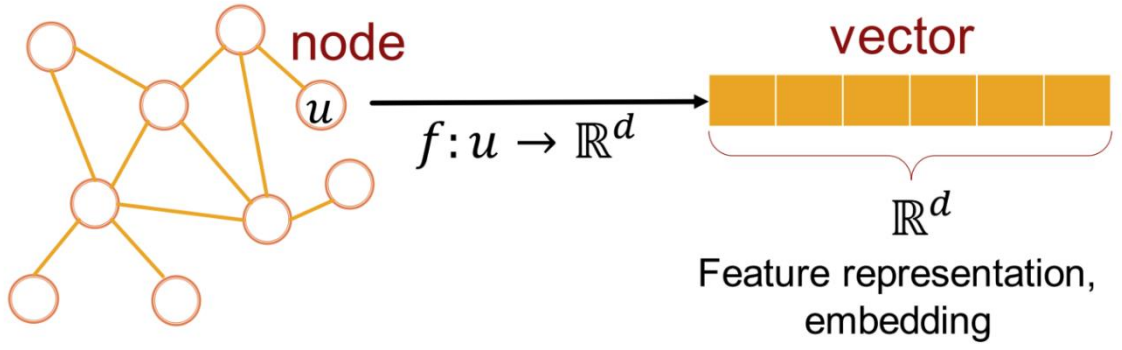
```
colors = ["#00C98D", "#5030C0", "#50F0F0"]
pos = nx.spring_layout(G)
lst_m = community.label_propagation_communities(G)
color_map_b = {}
keys = G.nodes()
values = "black"
for i in keys:
        color_map_b[i] = values
counter = 0
for c in lst_m:
  for n in c:
    color_map_b[n] = colors[counter]
  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()
```



Node Feature: Embeddings Overview

- 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

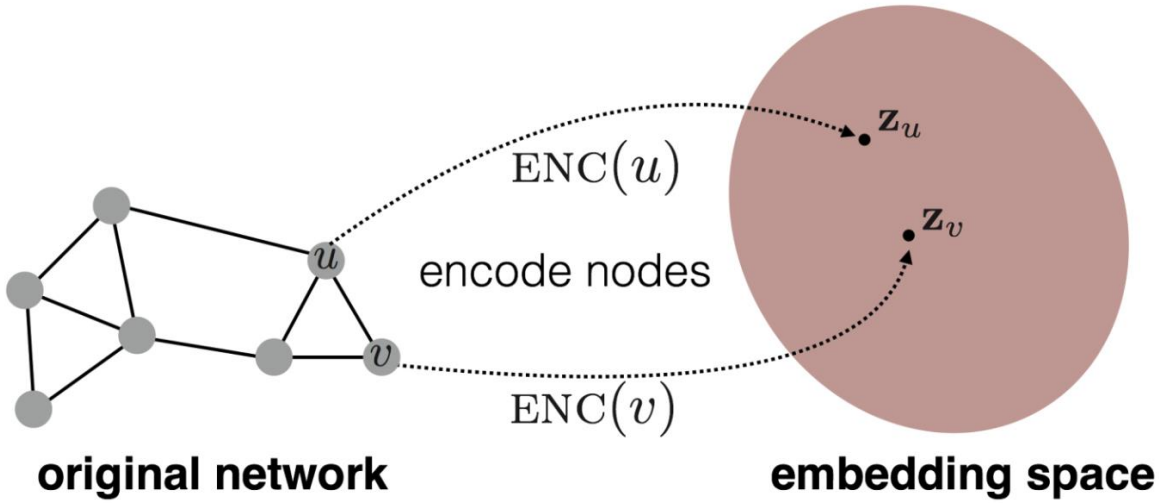


Node Feature: Node Embeddings

- Goal: Encode nodes → similarity in embedding space (dot product) ≈ similarity in the original graph.

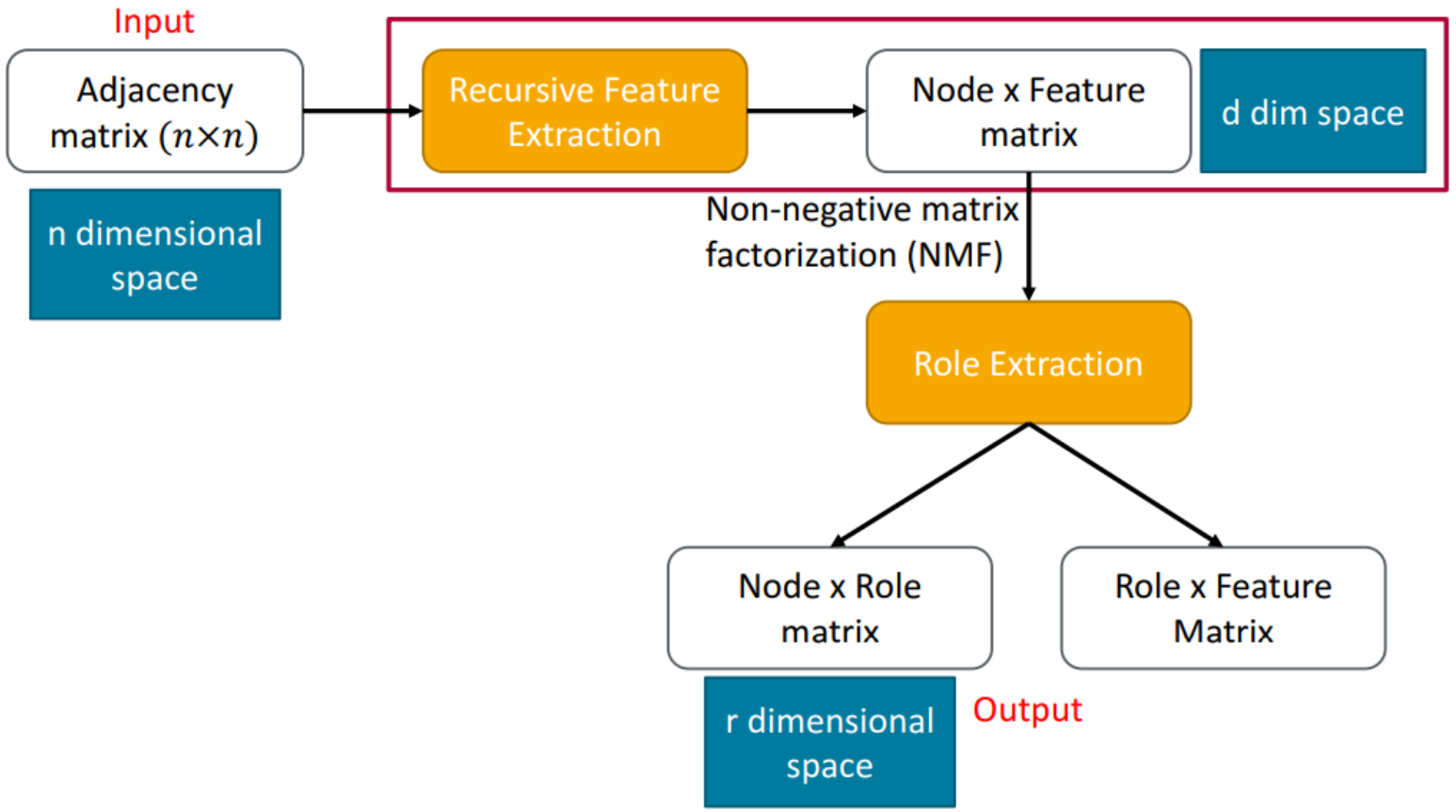
$$\underbrace{\text{similarity}(u, v)}_{\text{in the original network}} \approx \underbrace{\mathbf{z}_v^{\mathrm{T}} \mathbf{z}_u}_{\text{Similarity of the embedding}}$$

We need to define:
ENC(u)
Similarity(u, v)



RolX: Role eXtraction algorithm

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

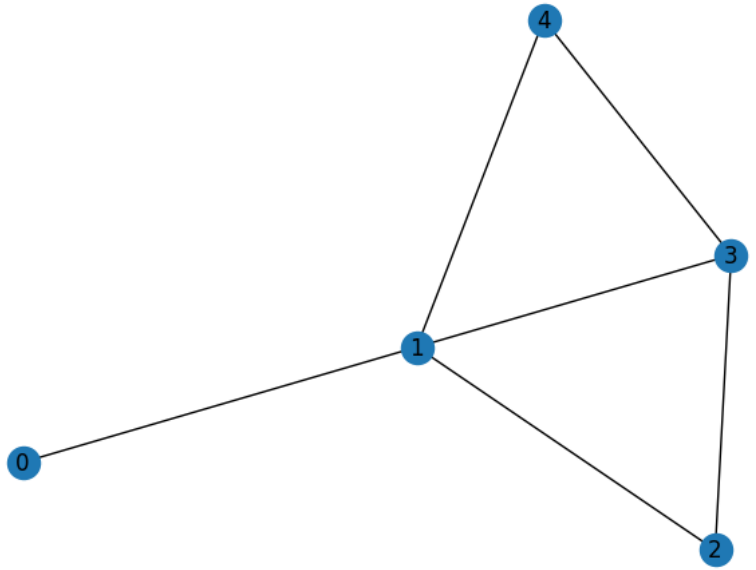


Henderson, K., Gallagher, B., Eliassi-Rad, T., Tong, H., Basu, S., Akoglu, L., Koutra, D., Faloutsos, C. and Li, L., 2012. Rolx: structural role extraction & mining in large graphs. SIGKDD

RolX: Role eXtraction algorithm: Sample code


```python
# 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))
```

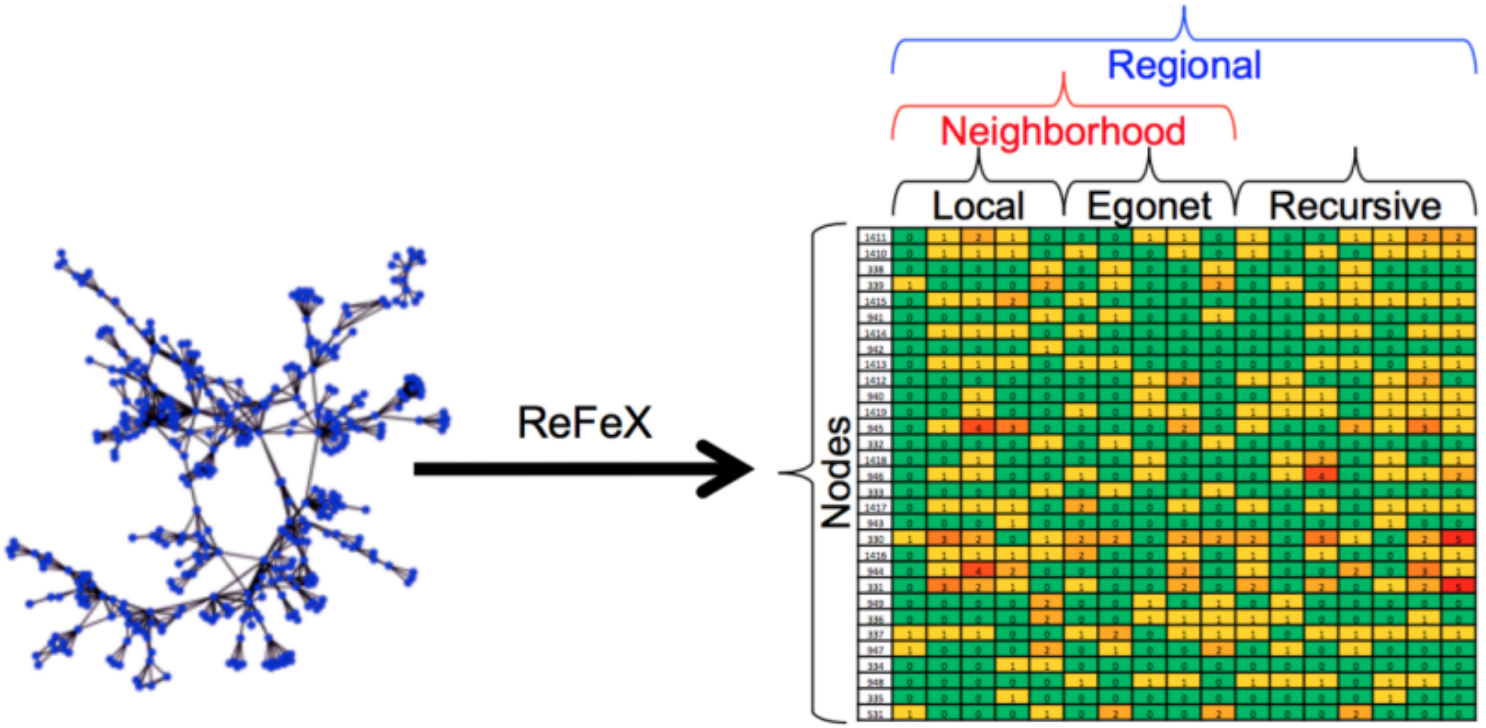
```
Node role assignments:
{0: 'role_1', 1: 'role_0', 2: 'role_1', 3: 'role_0', 4: 'role_1'}

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
```



Henderson, K., Gallagher, B., Eliassi-Rad, T., Tong, H., Basu, S., Akoglu, L., Koutra, D., Faloutsos, C. and Li, L., 2012 . Rolx: structural role extraction & mining in large graphs. SIGKDD

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)



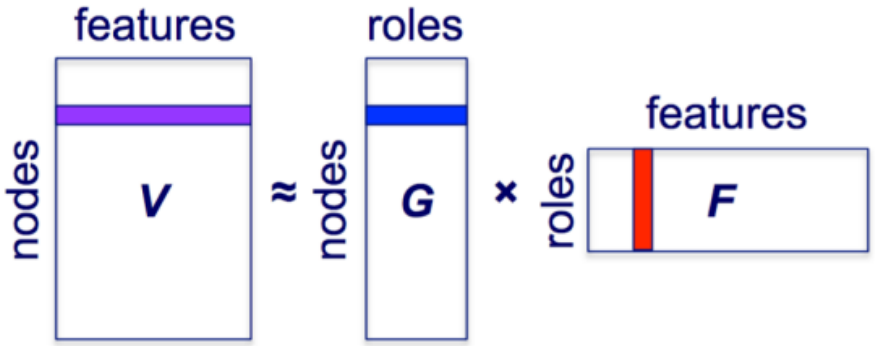
Henderson, K., Gallagher, B., Li, L., Akoglu, L., Eliassi-Rad, T., Tong, H. and Faloutsos, C., 2011. It's who you know: graph mining using recursive structural features. SIGKDD.

ReFeX: Mining features

- 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 x feature matrix V.
- Use non-negative matrix factorization: $V \approx GF$.



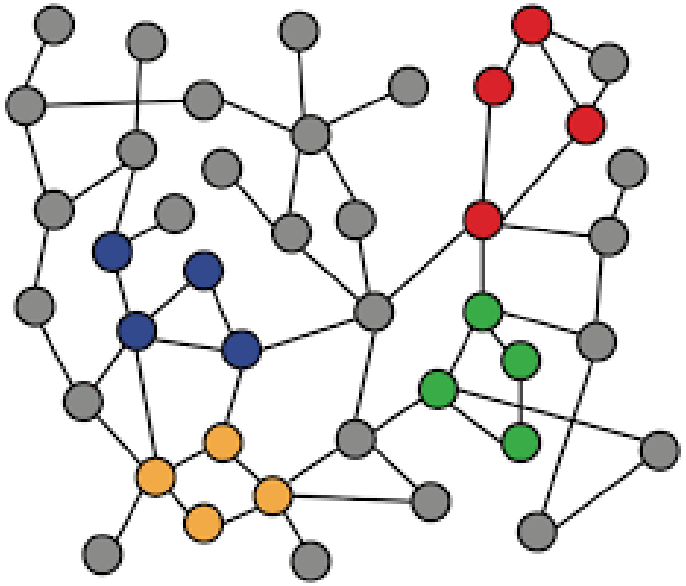
- The *G* matrix assigns nodes to roles.
- The *F* matrix represents how the features explain the roles.

Summary: Ways to Obtain Node Features

- 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.

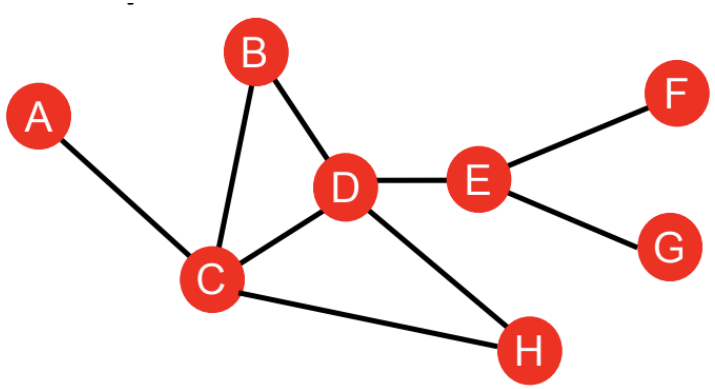
Edge Feature Extraction: Overview

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



Distance-based Features

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



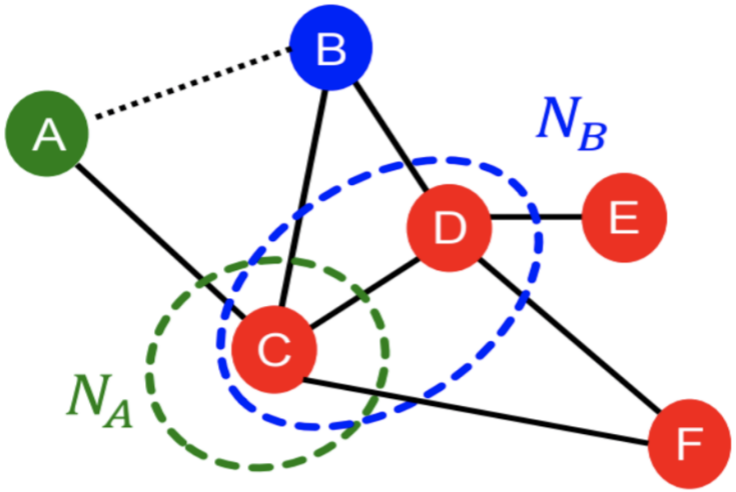
$$S_{BH} = S_{BE} = S_{AB} = 2$$

$$S_{BG} = S_{BF} = 3$$

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

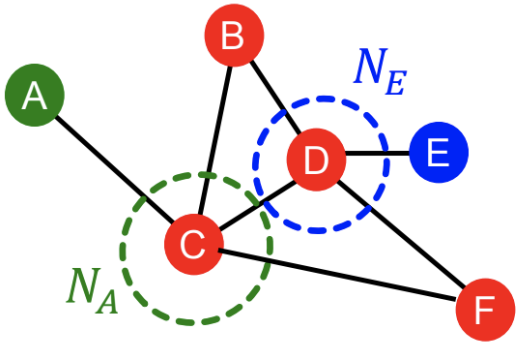
Local Neighborhood Overlap

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



Global Neighborhood Overlap

- Limitation of local neighborhood overlapping 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.



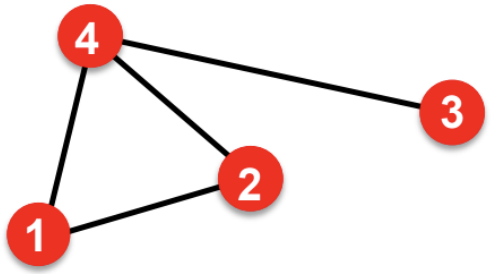
$$N_A \cap N_E = \phi$$
$$|N_A \cap N_E| = 0$$

→ Global neighborhood overlap features resolve the limitation by considering the entire graph.

Global Neighborhood Overlap: Katz Index

- 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



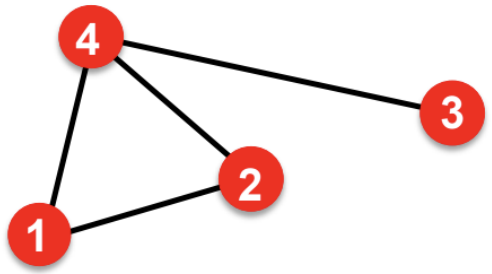
$$A = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

Ai,j = 1 if node i, j are connected

Katz Index

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

- $P_{uv}^{(1)}$ = #walks of length 1 (direct neighborhood) between u and v = A_{uv}

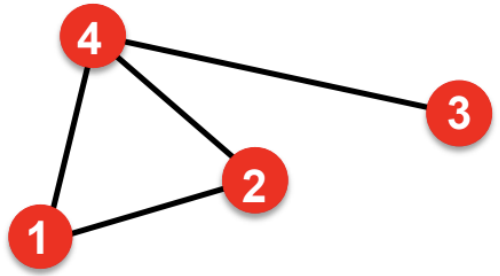


$$P_{12}^{(1)} = A_{12}$$

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

Ai,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

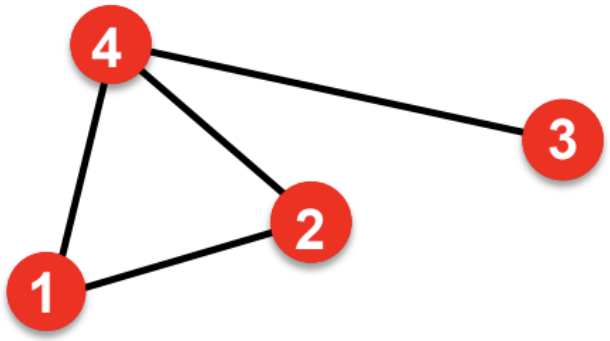
$P_{12}^{(2)} = A_{12}^2$

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

Power of adjacency

Katz Index

- 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 neighborhood) between u and v.
- A_{uv}^{2} specifies #walks of length 2 (neighbor of neighbor) between u and v.
- A_{uv}^{l} specifies #walks of length l.



Katz Index

- **Katz index** between v1 and v2 is calculated as **sum over all walk lengths.**

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

#walks of length l between v_1 and v_2

$0 < \beta < 1$: discount factor

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

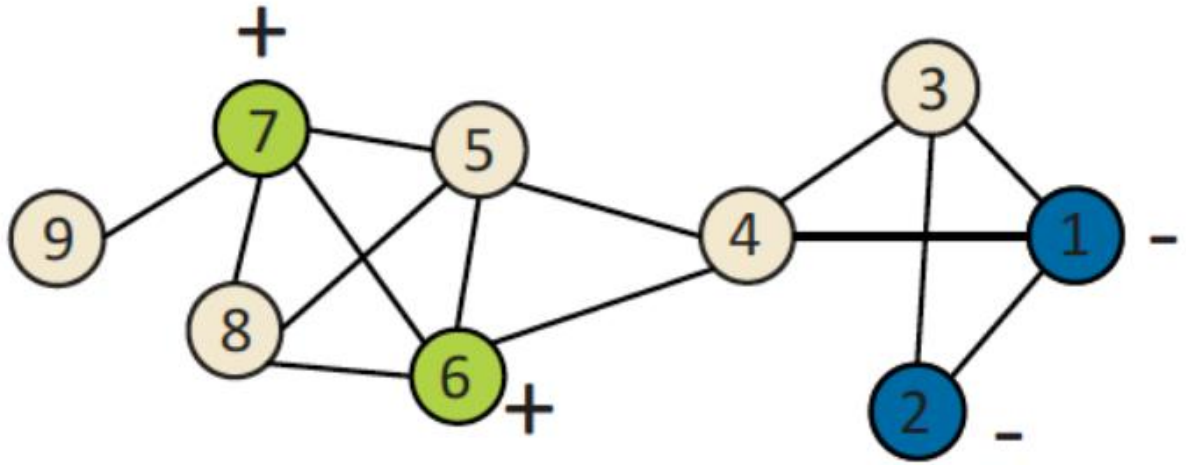
$$\boldsymbol{S} = \sum_{i=1}^{\infty} \beta^i \boldsymbol{A}^i = (\boldsymbol{I} - \beta \boldsymbol{A})^{-1} - \boldsymbol{I}$$

Summary: Ways to Obtain Edge Features

- Distance-based features:
  - Calculates shortest path between 2 nodes but cannot capture overlapping neighboods.
- Local neighborhood overlap:
  - Captures number of sharing neighborhoods between 2 nodes.
  - Only focuses nodes within 2-hop.
- Global neighborhood overlap:
  - Katz index uses entire graph structure to score 2 nodes.
  - It can capture the structure globally.

Classification with Network Data

- 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?

Why node classification?

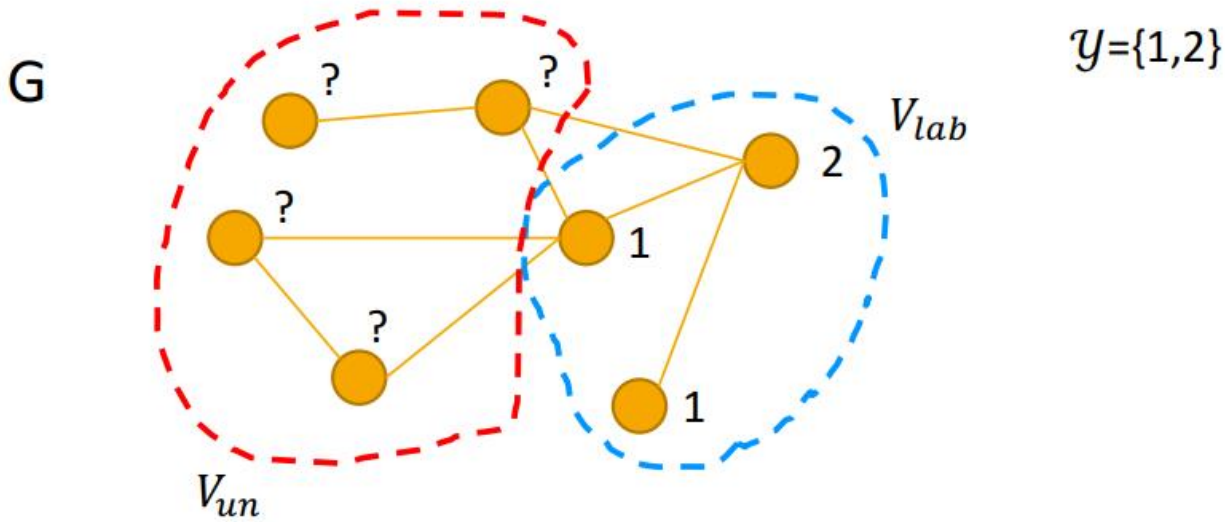
- Is this a friend or an acquittance?
- Recommendation systems to suggest objects (music, movies, activities)
- Automatically understand roles in a network (hubs, activators, influencing nodes, etc.)
- Identify experts for question answering systems
- Targeted advertising
- Study of communities (key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

Why node classification is useful?

- 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 \ V_{lab}
  - Y the set of m possible labels (e.g., Y={republican, democrat})
  - Y_{lab} = $\{y_1, y_2, \ldots, y_l\}$ the labels on labeled nodes in V_{lab}
- Problem:
  - Infer labels Y_{un} for all nodes in V_{un}

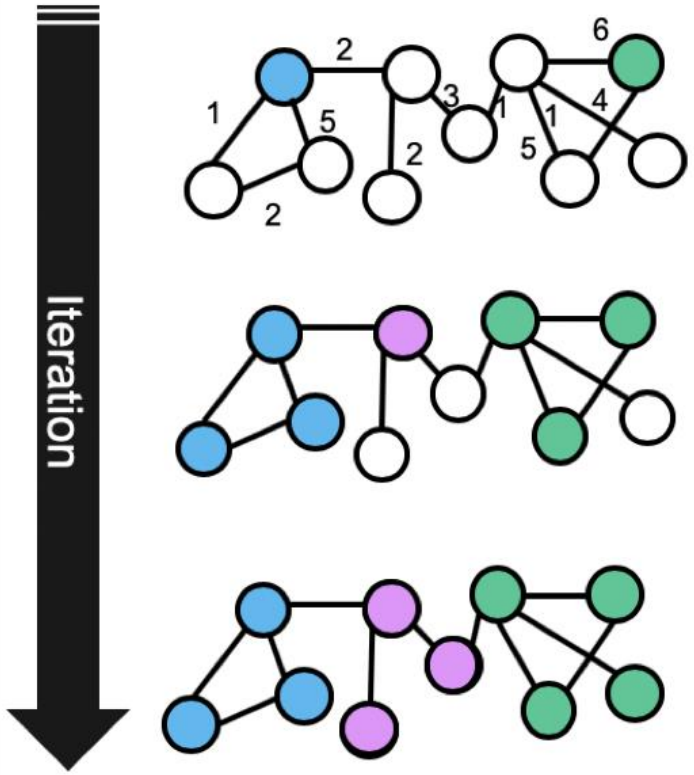


Node classification problem (2)

- 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 importance of the graph structure and node features

- 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.



Assumption: The label propagates on the network

The importance of the graph structure and node features

- Node features:
  - Measurable characteristics of the nodes that help
    - discriminating a node from another
    - or stating the similarity with other nodes.

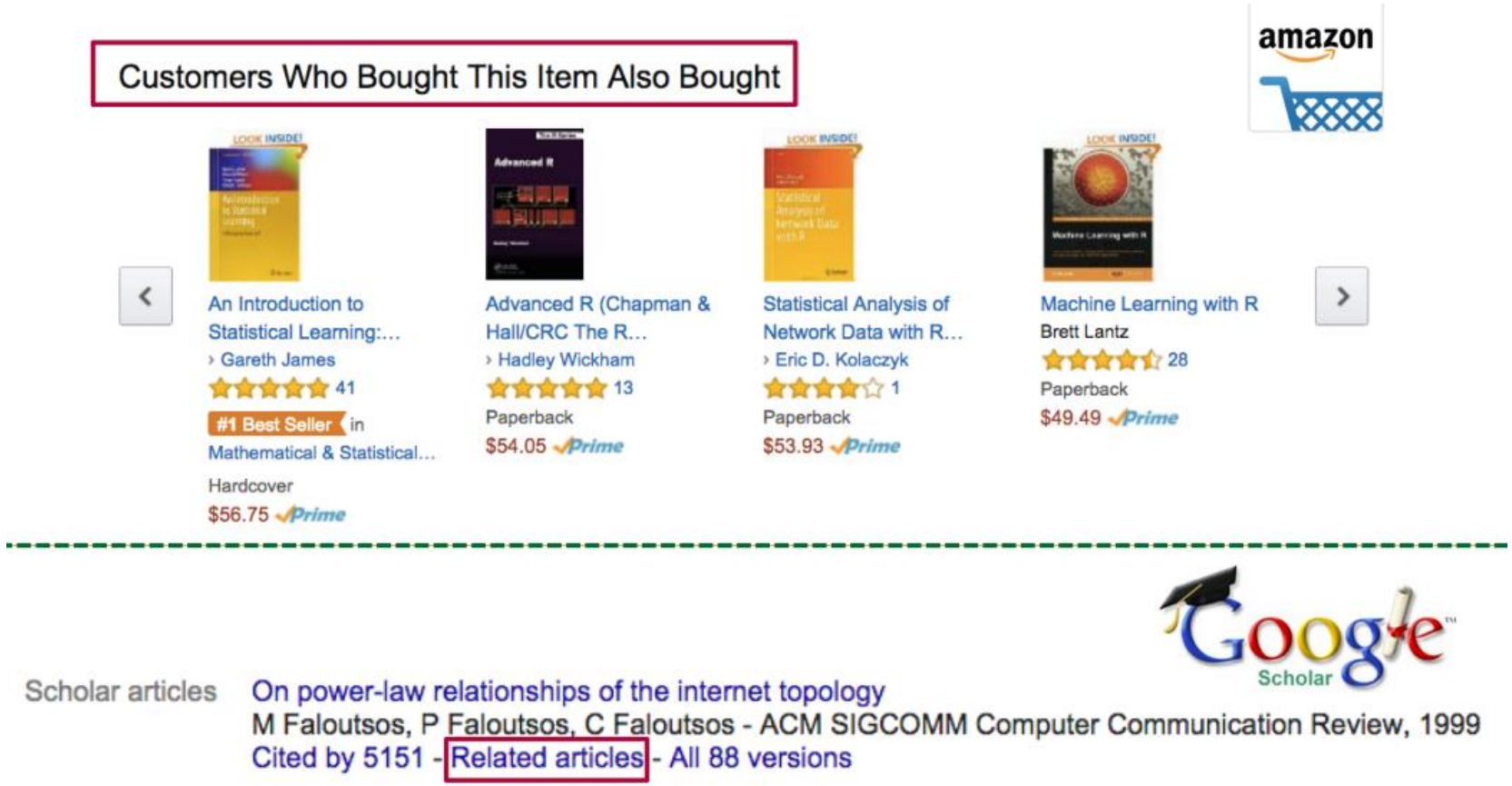
- Examples of features:
  - In/out degree of the node.
  - Number of L-labelled edges from that node.
  - Number of paths in that goes through the node.
  - Number of triangles.
  - Degree and number within ego-net edges.
  - etc.

Node classification approaches

- **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.

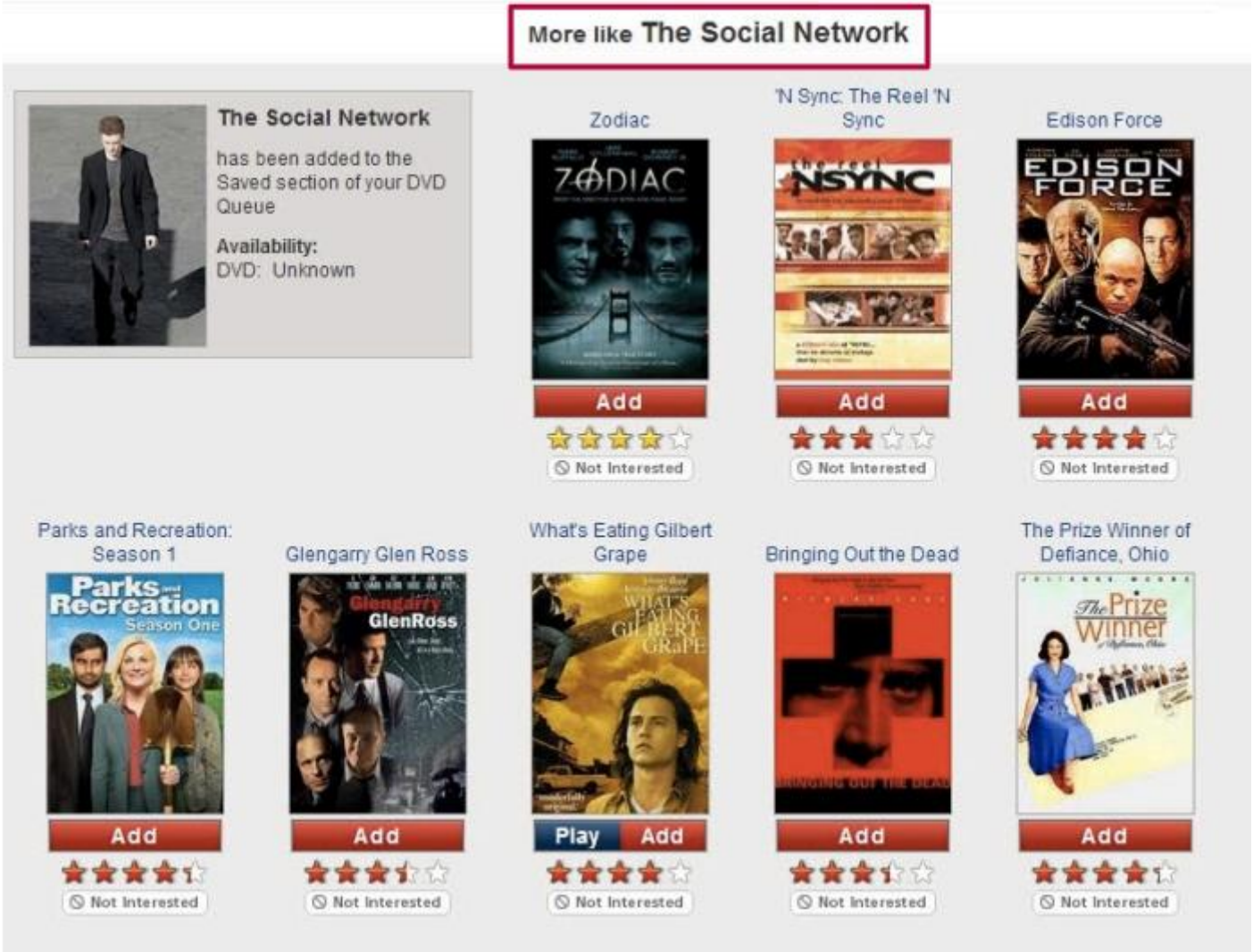
Similarity based

- Real-world Applications:



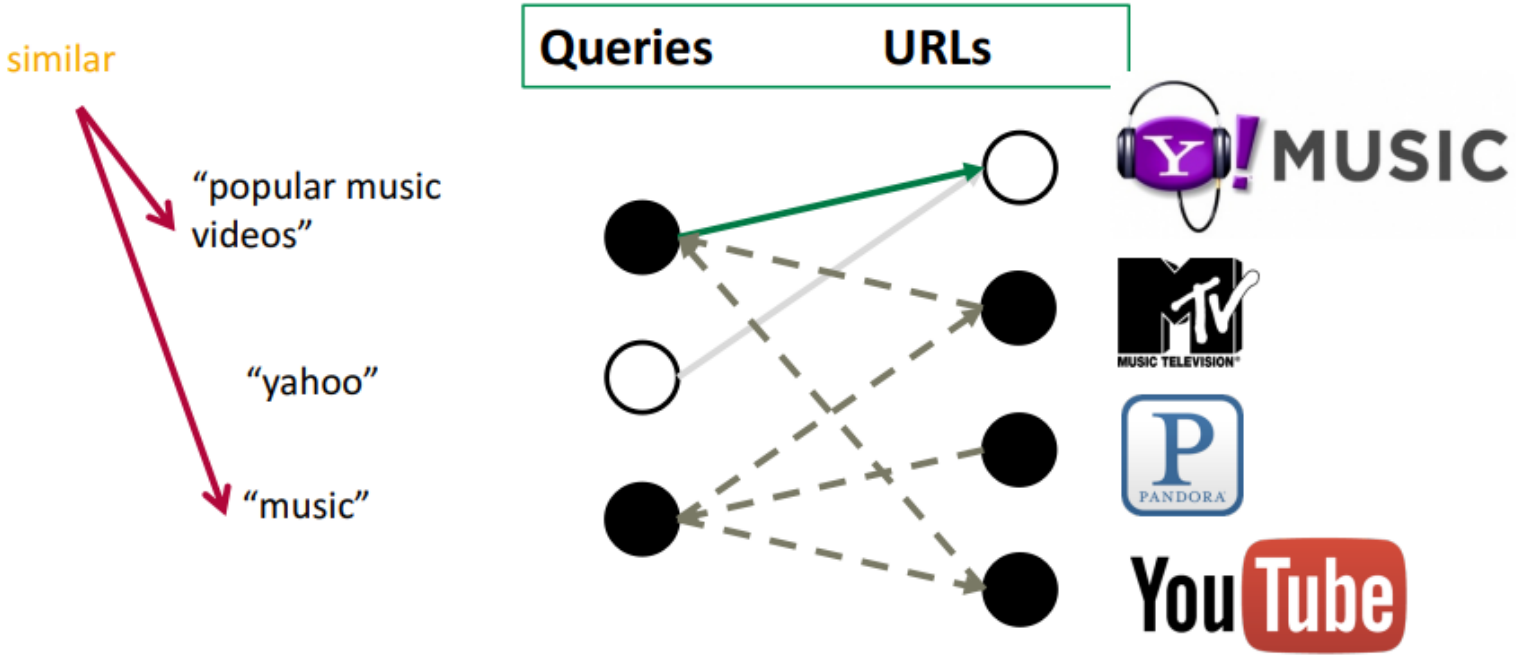
Similarity based

- Movies recommendations



Davide Mottin. Hasso Plattner Institute.

- 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.



Source: Google

History: Equivalences

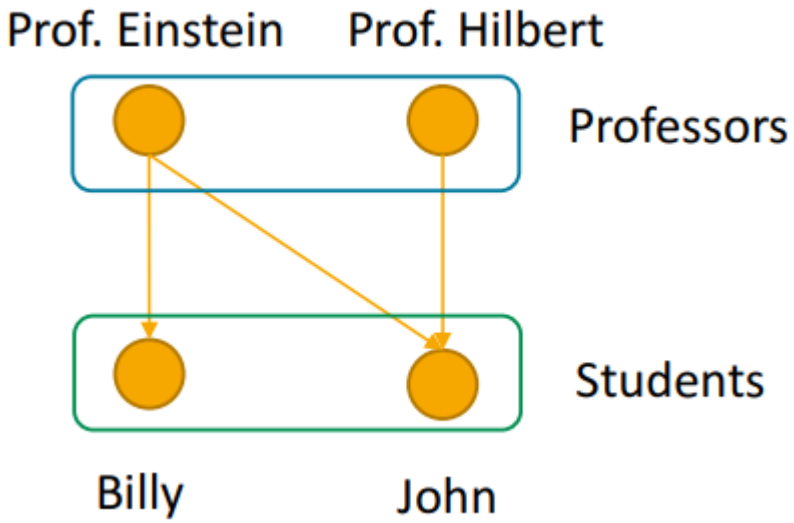
- Two nodes u and v are structurally equivalent if they have the same relationships to all other nodes.

- 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.

Regular equivalence

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

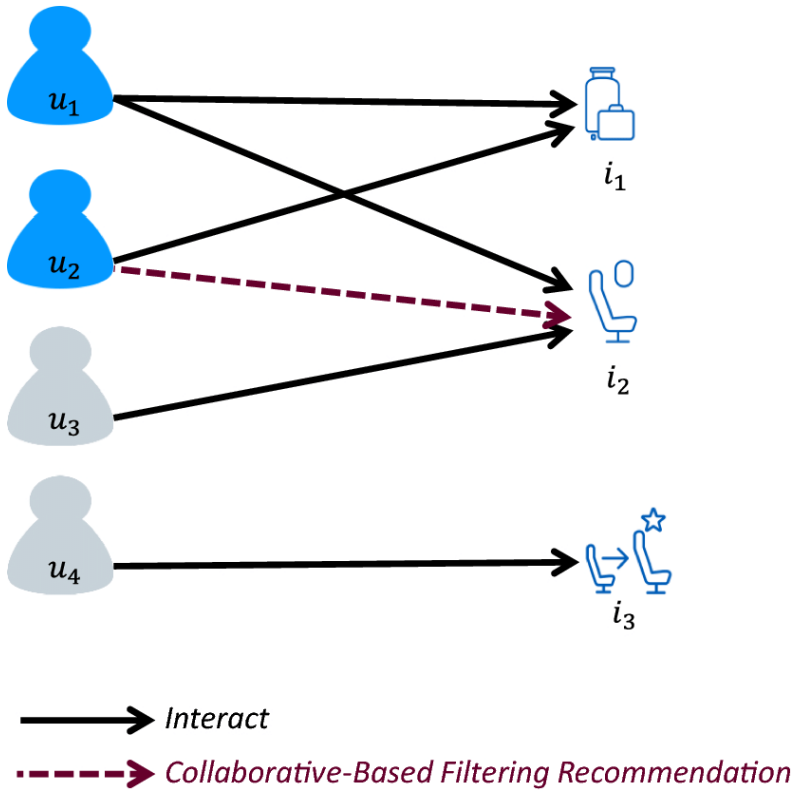


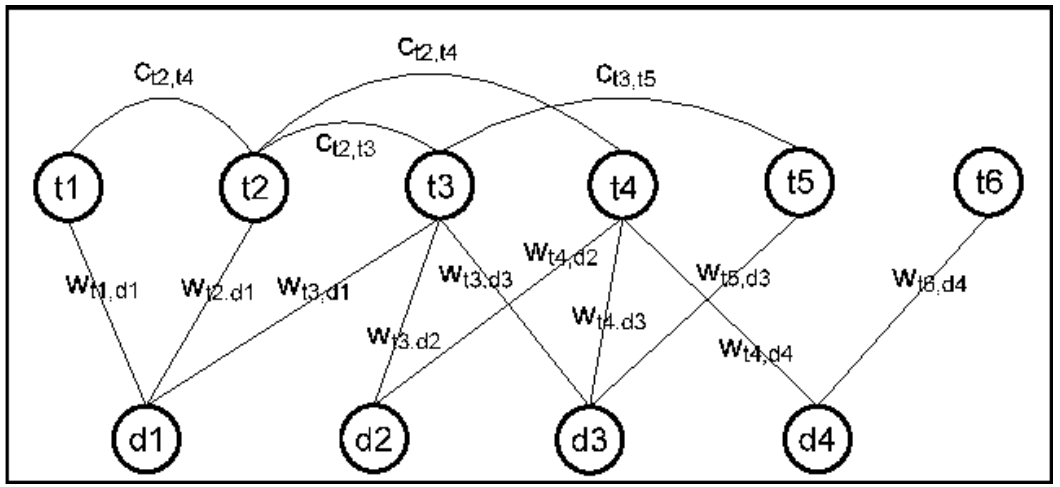
Billy and John are similar because they are both connected to a professor.
Same for prof. Einstein and Hilbert.

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

Regular equivalence

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

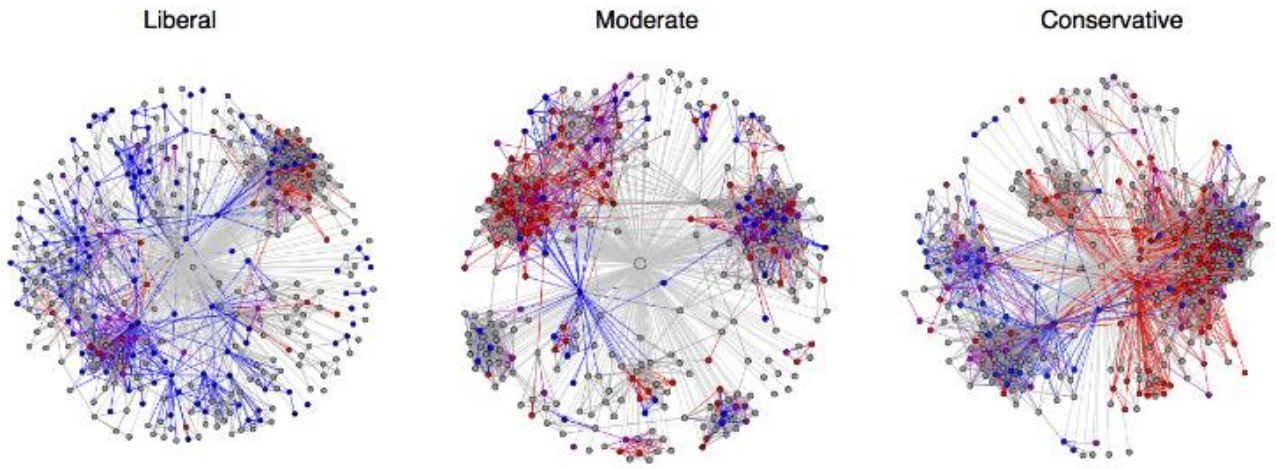




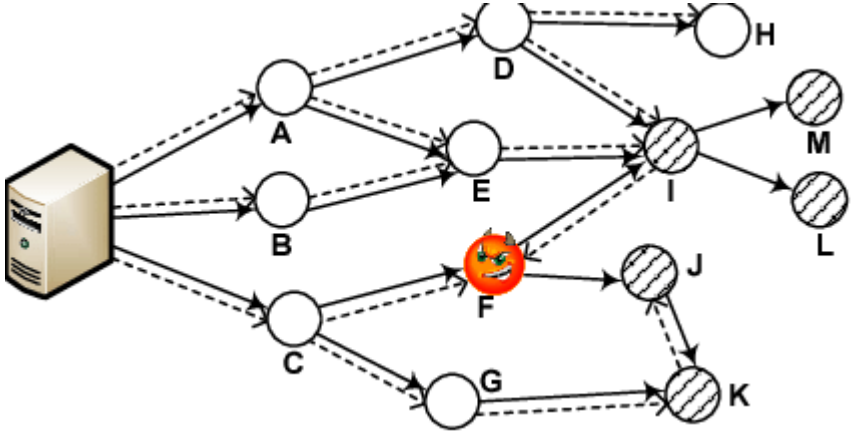
Source: Google

Application on Node Classification Tasks

- Social Network Analysis
  - Identifying user roles or community affiliations or politics group in social networks.



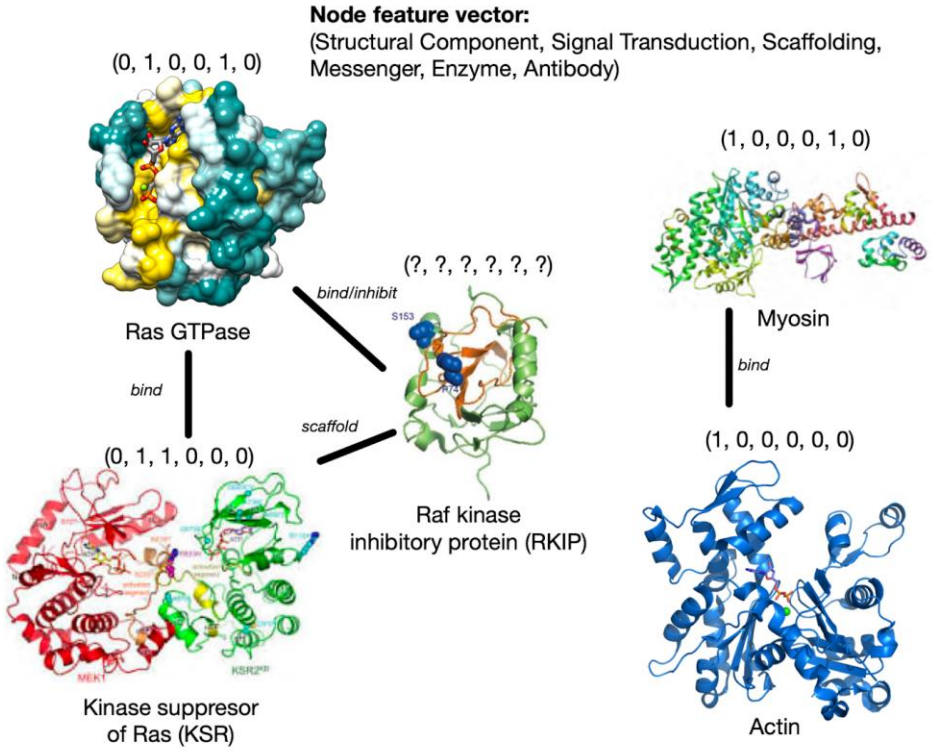
  - Detecting fake accounts or malicious users in online social networks.



https://snap-stanford.github.io/cs224w-notes/machine-learning-with-networks/message-passing-and-node-classification.

Application on Node Classification Tasks

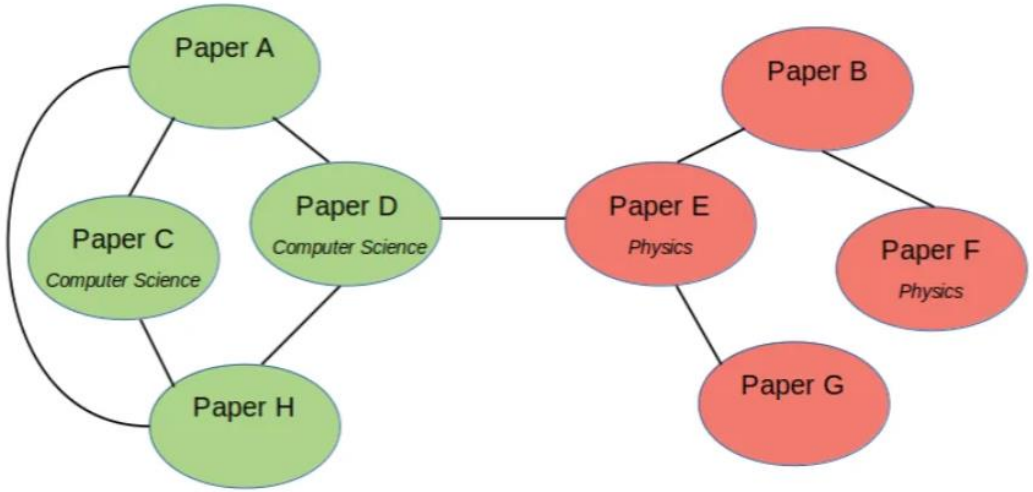
- Biological Networks:
  - Classify protein functions.
  - Identifying biomarkers or disease subtypes in biomedical networks for personalized medicine.

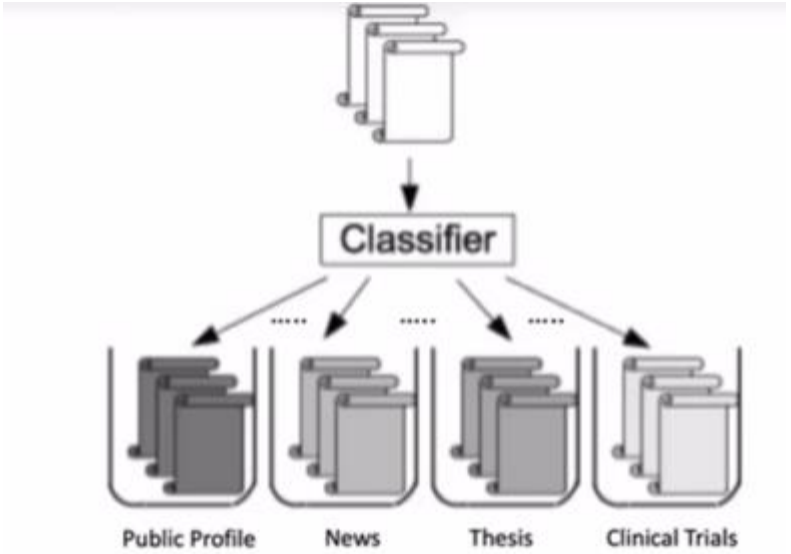


https://kevingyang.medium.com/protein-function-prediction-with-graph-neural-networks-e115ebff1939

Application on Node Classification Tasks

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

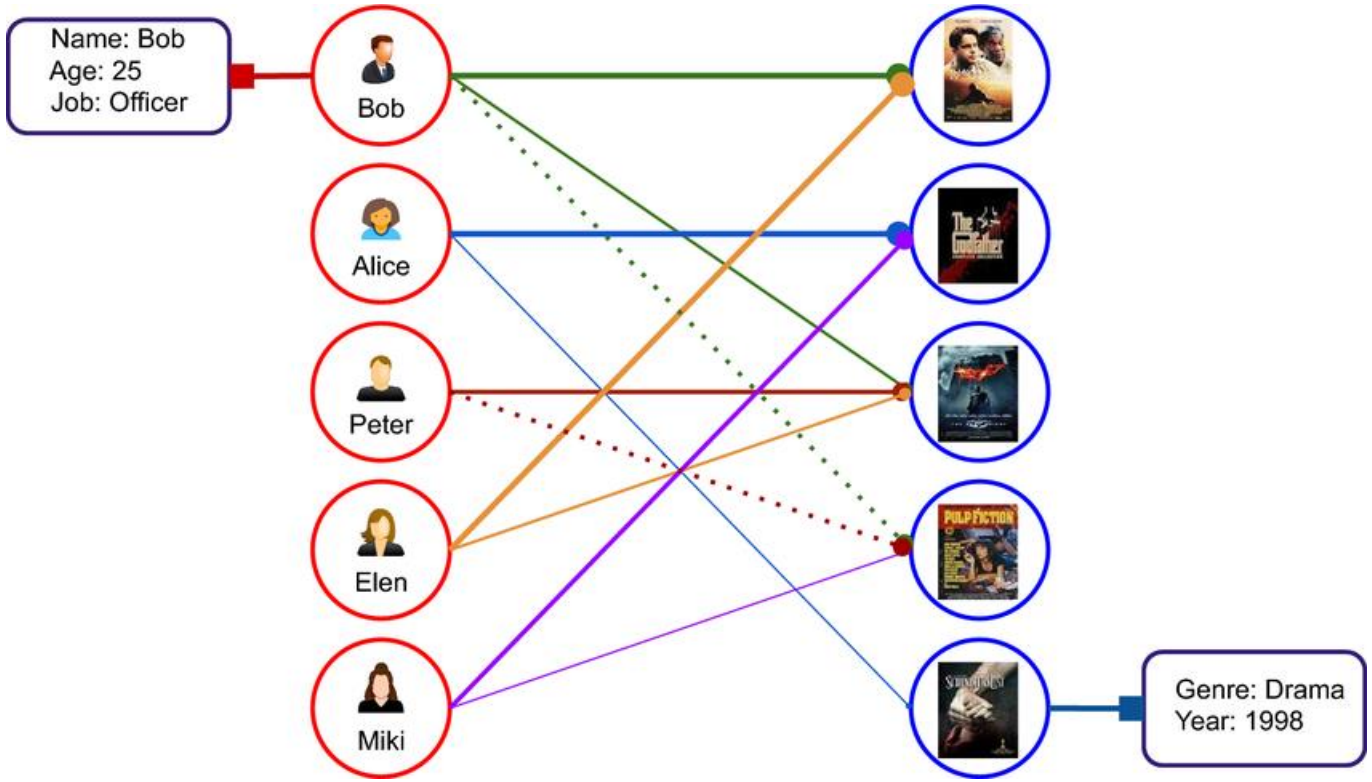




https://www.analyticsvidhya.com/blog/2023/03/how-to-classify-web-pages-using-machine-learning/

Application on Node Classification Tasks

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



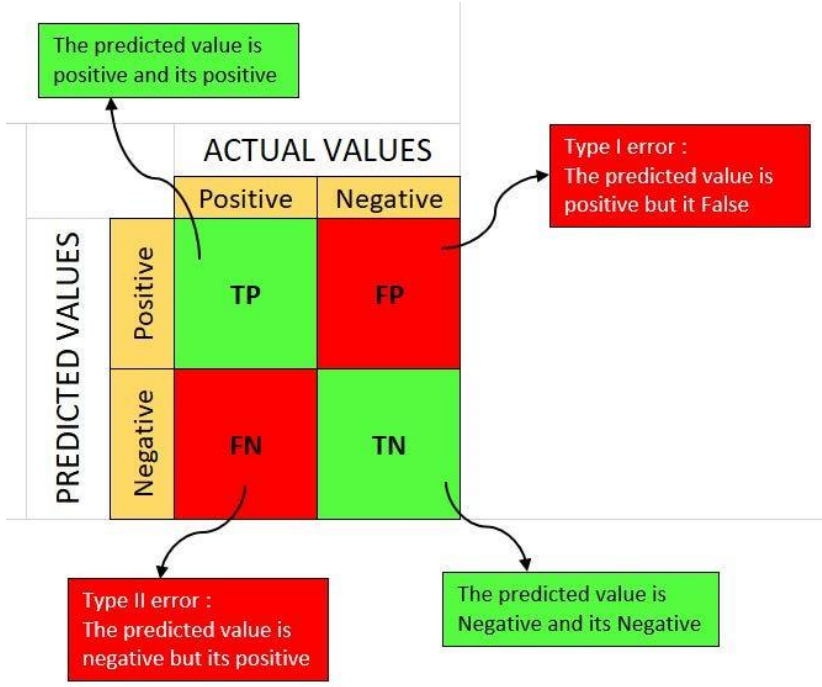
https://www.researchgate.net/publication/348345869_HeteGraph_graph_learning_in_recommender_systems_via_graph_convolutional_networks/figures?lo=1&utm_source=google&utm_medium=organic

Evaluation metrics

- Precision/Recall.
- Accuracy + weighted loss.
- ROC and AUC.

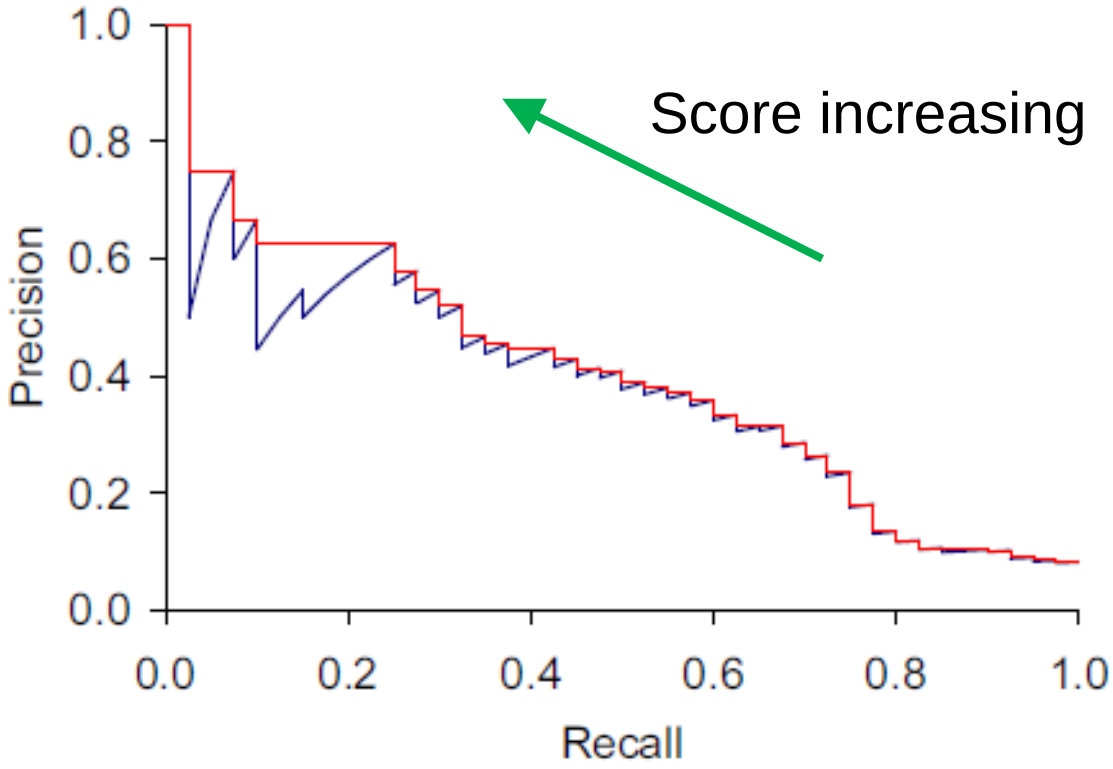
Precision and Recall

- 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.



Precision and Recall

- 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.



Be careful of “Accuracy”

- 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.

Weighted loss

- 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).

The weighted “F” measure

- A measure that naturally combines precision and recall is the β-weighted

F-measure:

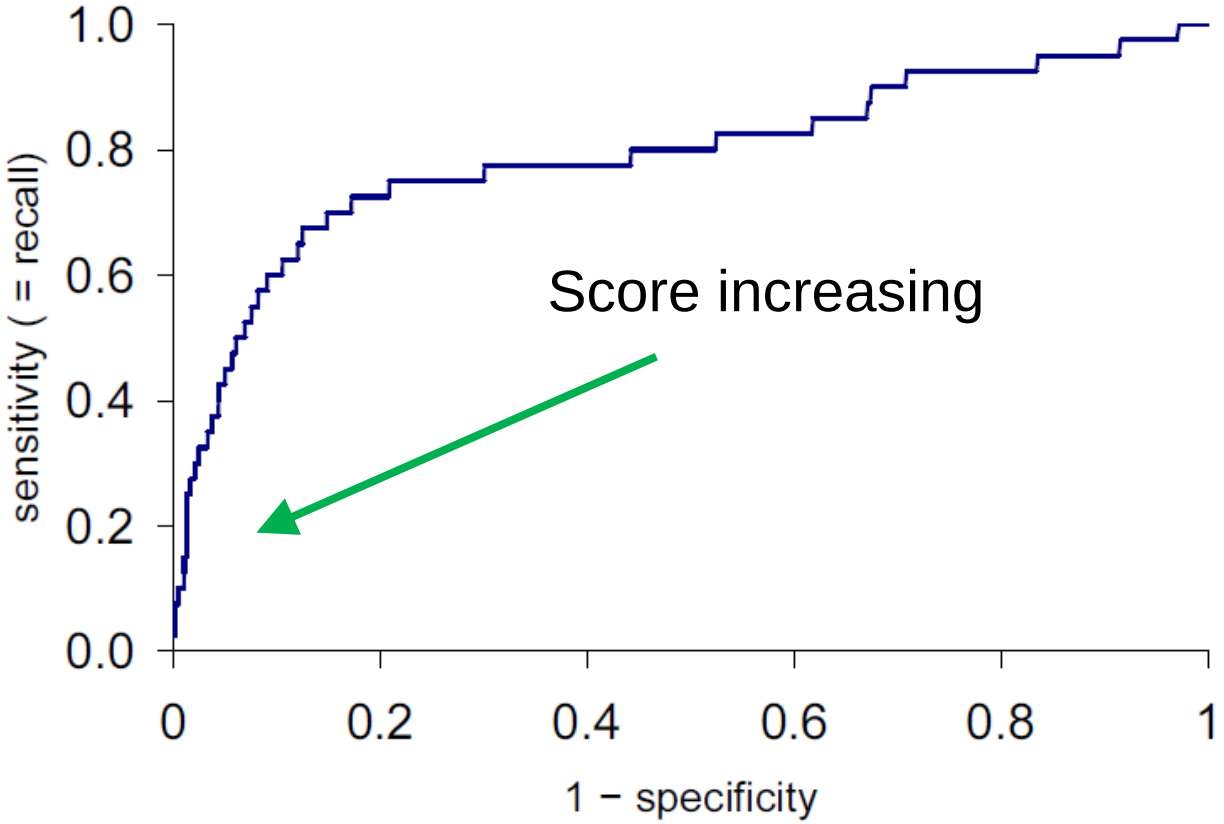
$$F = \frac{(\beta^2 + 1)PR}{\beta^2 P + R}$$

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

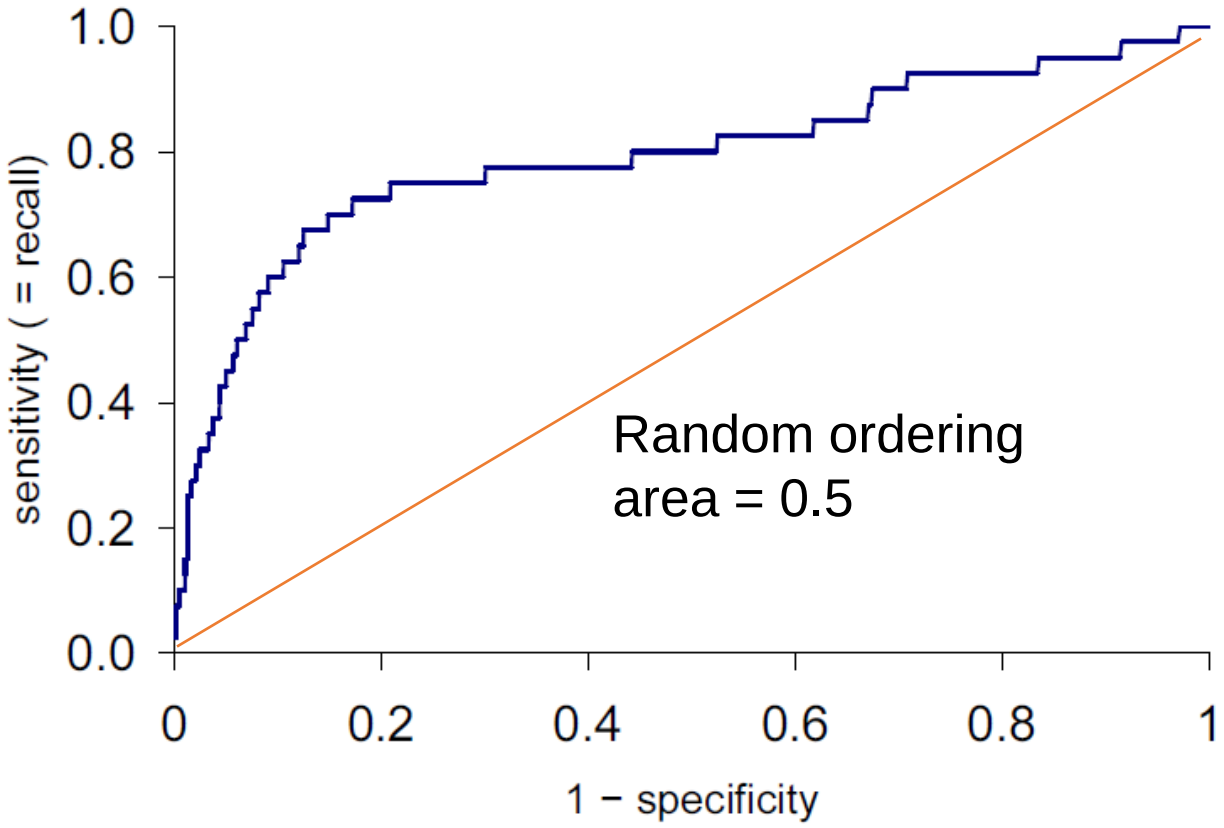
$$F_{\beta=1} = \frac{2PR}{P + R}$$

ROC plots

- 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 – 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