Ecole Nationale d'Electronique et des
Télécommunications de Sfax (ENET'COM)

Machine Learning with Graphs

3ème année IDSD-ID

Tarak BEN SAID
tarak.bensaid@enetcom.usf.tn

Machine Learning with Graphs

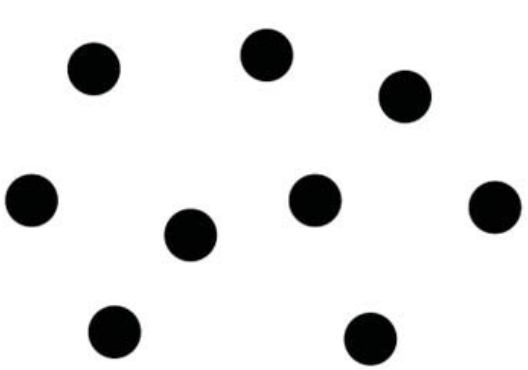
- Prerequisites
  - Machine Learning
  - Algorithms and graph theory
  - Probability and statistics
  - Python programming
- References
  - Graph Representation Learning by William L. Hamilton
  - Networks, Crowds, and Markets: Reasoning About a Highly Connected World by David Easley and Jon Kleinberg
  - Network Science by Albert-László Barabási
  - https://www.stanford.edu/

Machine Learning with Graphs

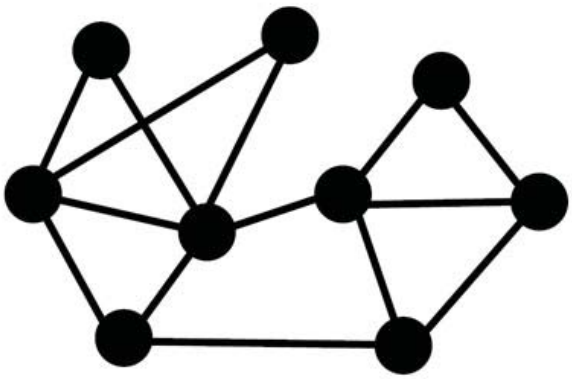
- Course Outline
  - Introduction
  - Traditional methods
  - Node embeddings
  - Link analysis
  - Label propagation for node classification
  - Graph Neural Networks
  - Applications of GNN

Introduction

- Why Graphs ?
  - Graphs are a general language for describing and analyzing entities with relations/interactions



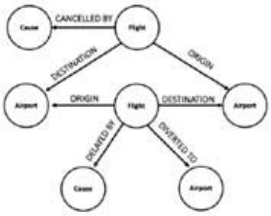
Isolated datapoints



Graph

Introduction

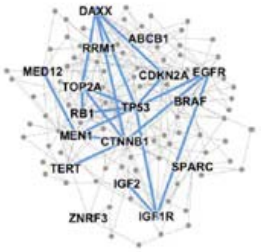
- Graph Data Types



Event Graphs

Image credit: SalientNetworks

Computer Networks



Disease Pathways

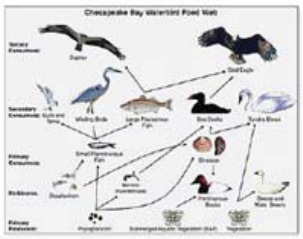


Image credit: Wikipedia

Food Webs

Image credit: Pinterest

Particle Networks

Image credit: visitlondon.com

Underground Networks

Introduction

- Graph Data Types

Image credit: Medium

Social Networks

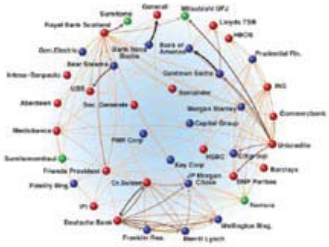
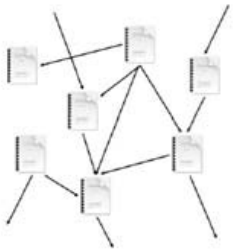


Image credit: Science

Economic Networks

Image credit: Lumen Learning

Communication Networks



Citation Networks

Image credit: Missoula Current News

Internet

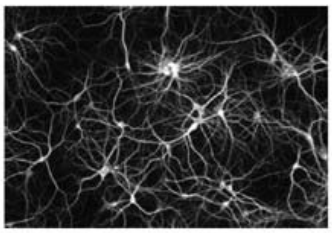


Image credit: The Conversation

Networks of Neurons

Introduction

- Graph Data Types

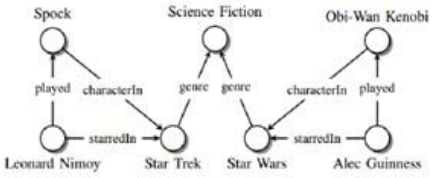


Image credit: Maximilian Nickel et al

Knowledge Graphs

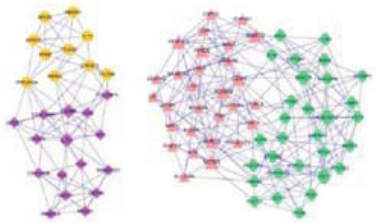


Image credit: ese.wustl.edu

Regulatory Networks

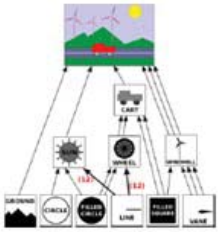


Image credit: math.hws.edu

Scene Graphs

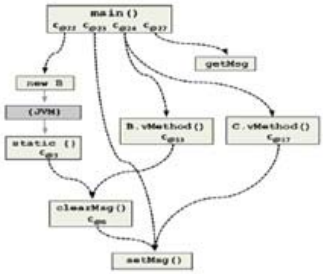


Image credit: ResearchGate

Code Graphs

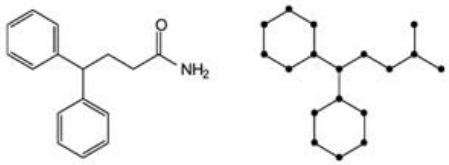


Image credit: MDPI

Molecules

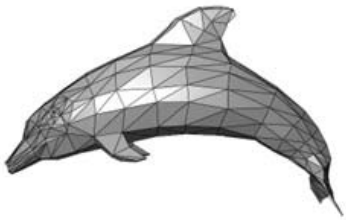
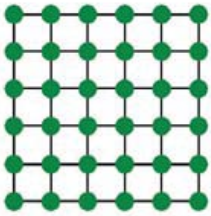


Image credit: Wikipedia

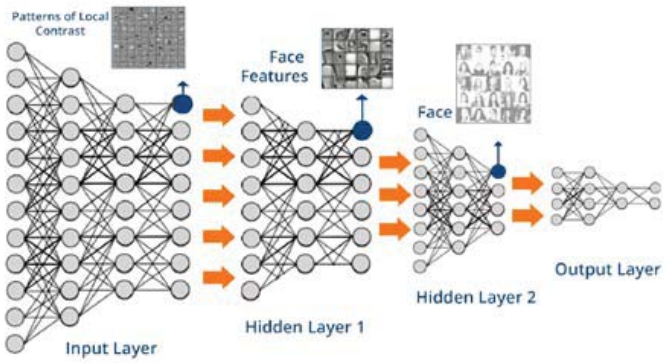
3D Shapes

Introduction

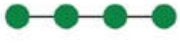
- Modern Deep Learning Toolbox
  - Simple data types
    - Sequences : text, speech,
    - Grid : images

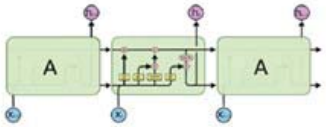


Images



Text/Speech

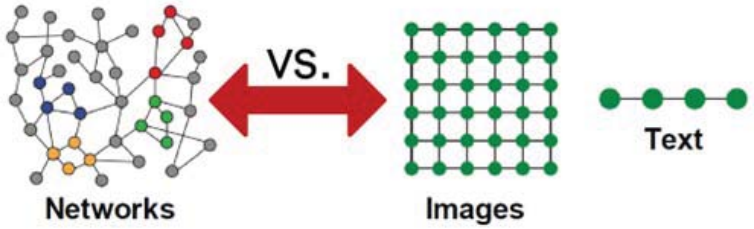




Introduction

- Modern Deep Learning Toolbox
  - Networks (Graphs) ?

    ➔ Networks are more complex
    - Arbitrary size and complex topological structure (no spatial locality like grids)

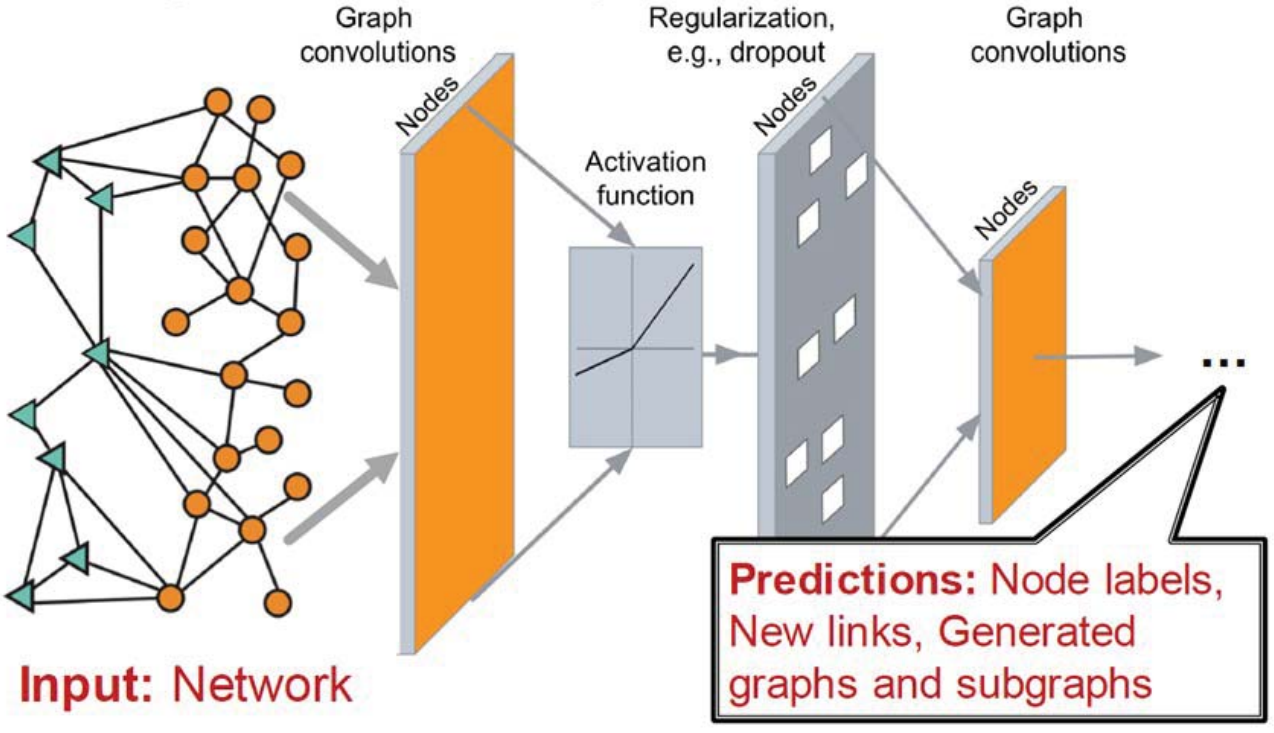


    - No fixed node ordering or reference point
    - Often dynamic and have multimodal features

  ➔ Graphs are the new frontier of deep learning

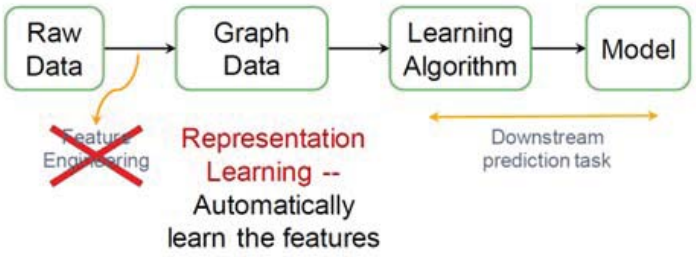
Introduction

- Deep Learning in Graphs

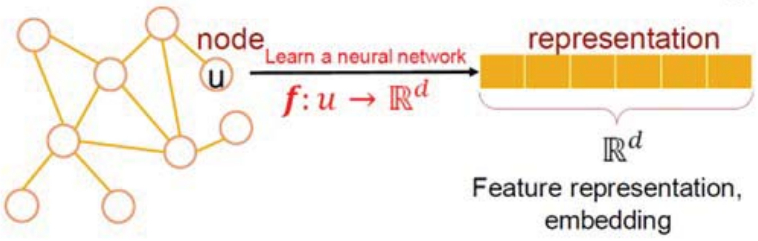


Introduction

- Representation Learning
  - No feature engineering : automatically learn features

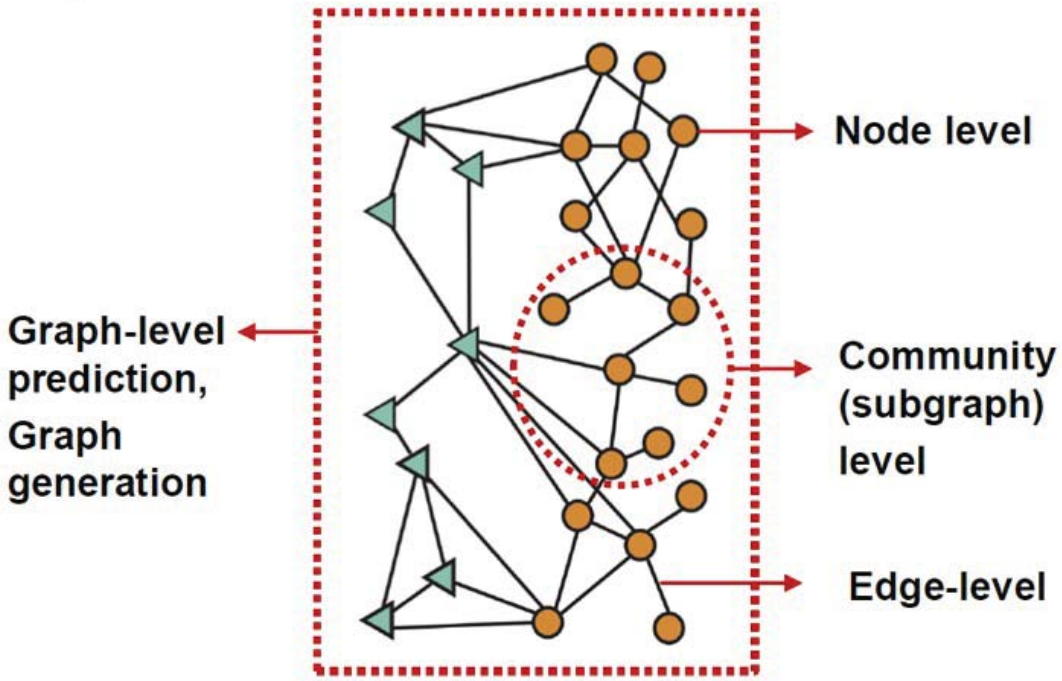


  - Map nodes to d-dimensional embeddings such that similar nodes in the network are embedded close together



Introduction

- Graph ML tasks

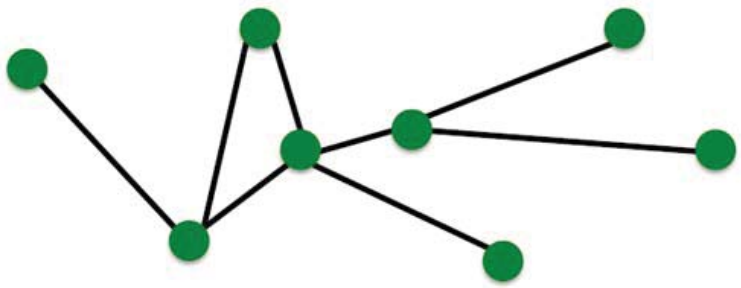


Introduction

- Graph ML tasks & applications
  - Node classification : predict a property of a node
    - Example : categorize online users /items
  - Link prediction : predict whether there are missing links
    - Example : knowledge graph completion, recommender system, drug side effects
  - Graph classification : categorize different graphs
    - Example : molecule property prediction
  - Clustering : detect if nodes form a community
    - Example : social circle detection
  - Graph generation (example : drug discovery)
  - Graph evolution (example : physical simulation)

Introduction

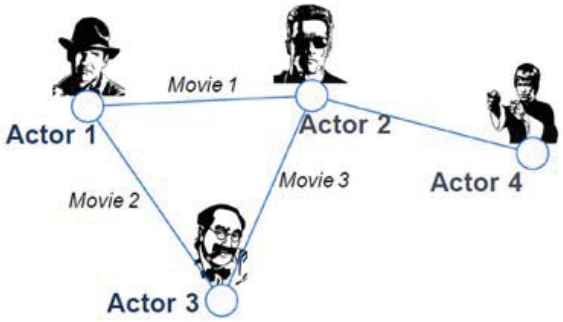
- Graph Representation
  - Components of a network

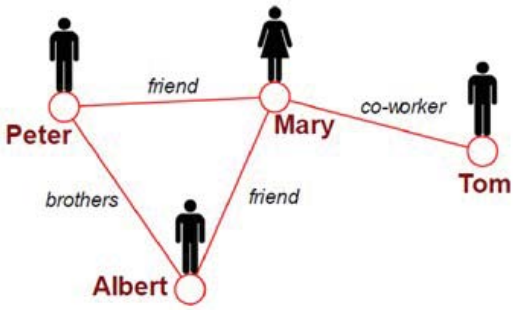


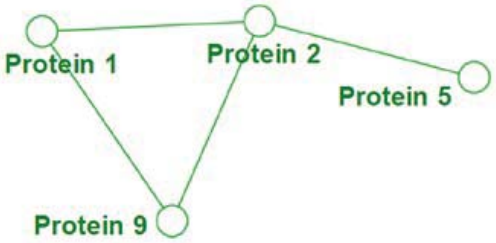
- **Objects:** nodes, vertices — N
- **Interactions:** links, edges — E
- **System:** network, graph — $G(N,E)$

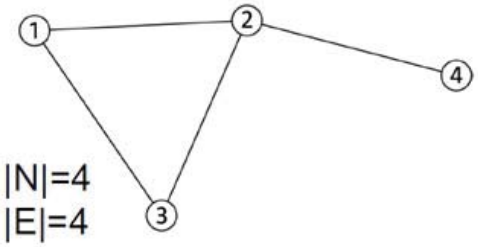
Introduction

- Graph Representation
  - Graphs : A Common Language







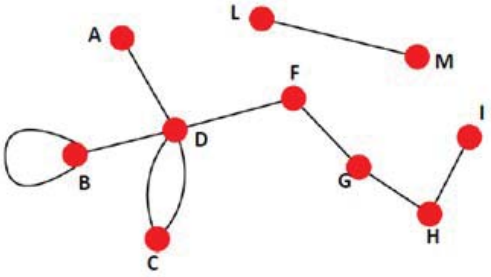


Introduction

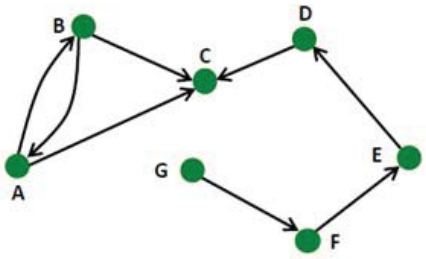
- Graph Representation
  - How to build a graph
    - What are nodes ?
    - What are edges ?
  - The choice of the proper network representation of a given domain/problem determines our ability to use networks successfully
    - In some cases, there is a unique, unambiguous representation
    - In other cases, the representation is by no means unique
  - The way you assign links will determine the nature of the question you can study

Introduction

- Graph properties
  - Undirected vs Directed
    - **Links:** undirected (symmetrical, reciprocal)



    - **Examples:**
      - Collaborations
      - Friendship on Facebook
    - **Links:** directed (arcs)

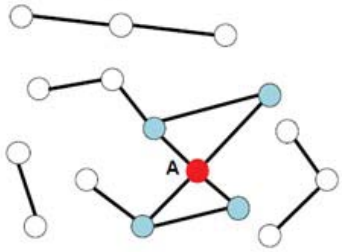


    - **Examples:**
      - Phone calls
      - Following on Twitter

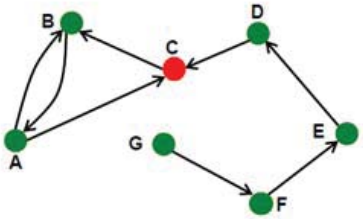
Introduction

- Graph properties
  - Node degrees

Undirected



Directed



Source: Node with $k^{in} = 0$
Sink: Node with $k^{out} = 0$

Node degree, k_i: the number of edges adjacent to node ***i***

$$k_A = 4$$

Avg. degree: $\overline{k} = \langle k \rangle = \frac{1}{N} \sum_{i=1}^{N} k_i = \frac{2E}{N}$

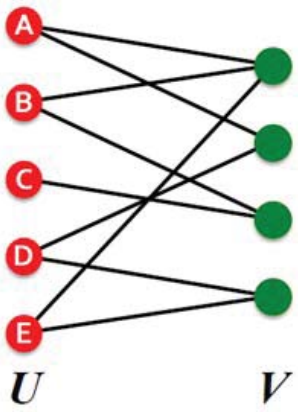
In directed networks we define an **in-degree** and **out-degree.**
The (total) degree of a node is the sum of in- and out-degrees.

$$k_C^{in} = 2 \qquad k_C^{out} = 1 \qquad k_C = 3$$

$$\overline{k} = \frac{E}{N} \qquad \overline{k^{in}} = \overline{k^{out}}$$

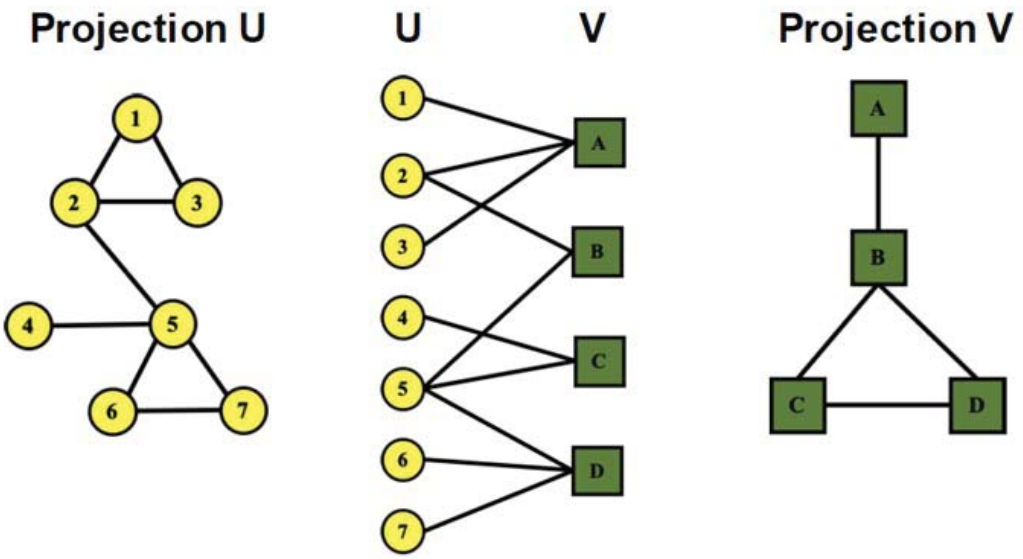
Introduction

- Graph properties
  - Bipartite graph
    - A bipartite graph is a graph whose nodes can be divided into two disjoint sets ***U*** and ***V*** such that every link connects a node in ***U*** to one in ***V***; that is, ***U*** and ***V*** are independent sets
    - Examples:
      - Authors-to-Papers
      - Actors-to-Movies
      - Users-to-Movies



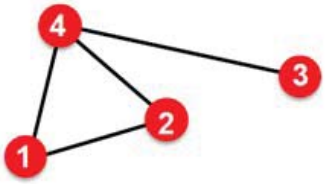
Introduction

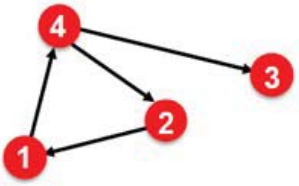
- Graph properties
  - Bipartite graph
    - Projected/Folded Bipartite Graph



Introduction

- Graph properties
  - Adjacency Matrix





$A_{ij} = 1$ if there is a link from node i to node j

$A_{ij} = 0$ otherwise

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

Note that for a directed graph (right) the matrix is not symmetric.

Sparse

Introduction

- Graph properties
  - Adjacency Matrix

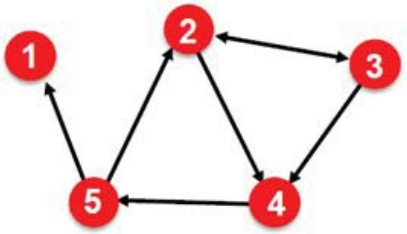
Most real-world networks are sparse

$E << E_{max}$ (or $k << N-1$)

NETWORK	NODES	LINKS	DIRECTED/ UNDIRECTED	N	L	<k>
Internet	Routers	Internet connections	Undirected	192,244	609,066	6.33
WWW	Webpages	Links	Directed	325,729	1,497,134	4.60
Power Grid	Power plants, transformers	Cables	Undirected	4,941	6,594	2.67
Phone Calls	Subscribers	Calls	Directed	36,595	91,826	2.51
Email	Email Addresses	Emails	Directed	57,194	103,731	1.81
Science Collaboration	Scientists	Co-authorship	Undirected	23,133	93,439	8.08
Actor Network	Actors	Co-acting	Undirected	702,388	29,397,908	83.71
Citation Network	Paper	Citations	Directed	449,673	4,689,479	10.43
E. Coli Metabolism	Metabolites	Chemical reactions	Directed	1,039	5,802	5.58
Protein Interactions	Proteins	Binding interactions	Undirected	2,018	2,930	2.90

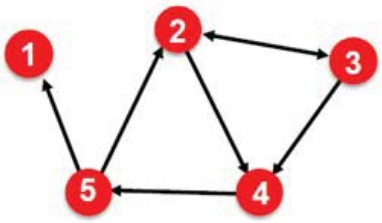
Introduction

- Graph properties
  - Edge List (representation)
    - (2, 3)
    - (2, 4)
    - (3, 2)
    - (3, 4)
    - (4, 5)
    - (5, 2)
    - (5, 1)



Introduction

- Graph properties
  - Adjacency List (representation)
    - Easier to work with if network is
      - Large
      - Sparse
    - Allows us to quickly retrieve all neighbors of a given node
      - 1:
      - 2: 3, 4
      - 3: 2, 4
      - 4: 5
      - 5: 1, 2

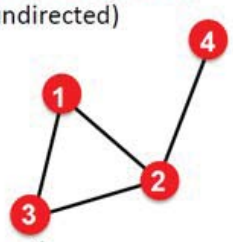


Introduction

- Graph properties
  - Node and Edge Attributes
    - Weight (e.g., frequency of communication)
    - Ranking (best friend, second best friend…)
    - Type (friend, relative, co-worker)
    - Sign : friend vs Foe, Trust vs Distrust
    - Properties depending on the structure of the rest of the graph : Number of common friends

Introduction

- Graph properties
  - Node and Edge Attributes
    - **Unweighted**
      (undirected)



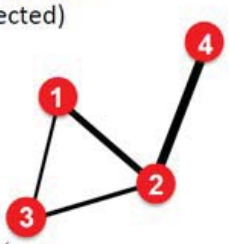
$$A_{ij} = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

$$A_{ii} = 0 \qquad A_{ij} = A_{ji}$$

$$E = \frac{1}{2}\sum_{i,j=1}^{N} A_{ij} \qquad \bar{k} = \frac{2E}{N}$$

Examples: Friendship, Hyperlink

    - **Weighted**
      (undirected)



$$A_{ij} = \begin{pmatrix} 0 & 2 & 0.5 & 0 \\ 2 & 0 & 1 & 4 \\ 0.5 & 1 & 0 & 0 \\ 0 & 4 & 0 & 0 \end{pmatrix}$$

$$A_{ii} = 0 \qquad A_{ij} = A_{ji}$$

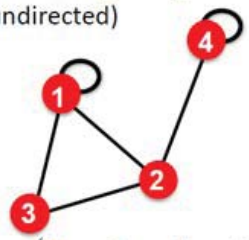
$$E = \frac{1}{2}\sum_{i,j=1}^{N} nonzero(A_{ij}) \qquad \bar{k} = \frac{2E}{N}$$

Examples: Collaboration, Internet, Roads

Introduction

- Graph properties
  - Self-loops and Multigraph
    - **Self-edges (self-loops)**

(undirected)



$$A_{ij} = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}$$

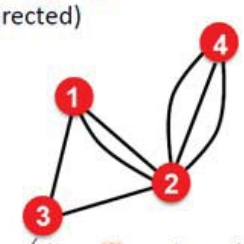
$A_{ii} \neq 0$ $\quad A_{ij} = A_{ji}$

$$E = \frac{1}{2}\sum_{i,j=1, i\neq j}^{N} A_{ij} + \sum_{i=1}^{N} A_{ii}$$

Examples: Proteins, Hyperlinks

    - **Multigraph**

(undirected)



$$A_{ij} = \begin{pmatrix} 0 & 2 & 1 & 0 \\ 2 & 0 & 1 & 3 \\ 1 & 1 & 0 & 0 \\ 0 & 3 & 0 & 0 \end{pmatrix}$$

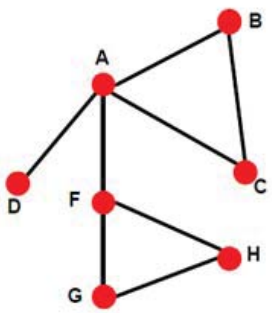
$A_{ii} = 0$ $\quad A_{ij} = A_{ji}$

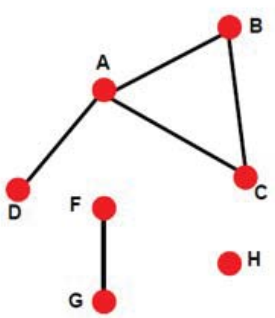
$$E = \frac{1}{2}\sum_{i,j=1}^{N} nonzero(A_{ij}) \qquad \bar{k} = \frac{2E}{N}$$

Examples: Communication, Collaboration

Introduction

- Graph properties
  - Connectivity of undirected graph
    - A graph is connected if any two nodes can be joined by a path
    - A disconnected graph is made up by two or more connected components





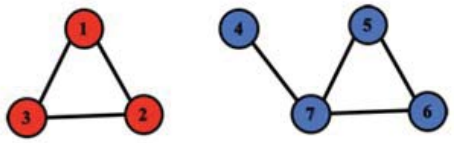
Largest Component:
Giant Component

Isolated node (node H)

Introduction

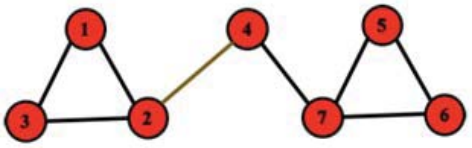
- Graph properties
  - Connectivity of undirected graph
    - The adjacency matrix of a network with several components can be written in a block-diagonal form, so that nonzero elements are confined to squares, with all other elements being zero

Disconnected



0	1	1	0	0	0	0
1	0	1	0	0	0	0
1	1	0	0	0	0	0
0	0	0	0	0	0	1
0	0	0	0	0	1	1
0	0	0	0	1	0	1
0	0	0	1	1	1	0

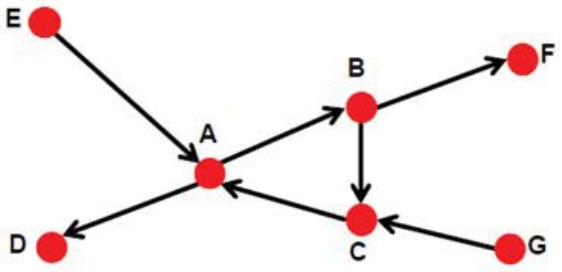
Connected



0	1	1	0	0	0	0
1	0	1	1	0	0	0
1	1	0	0	0	0	0
0	1	0	0	0	0	1
0	0	0	0	0	1	1
0	0	0	0	1	0	1
0	0	0	1	1	1	0

Introduction

- Graph properties
  - Connectivity of directed graph
    - Strongly connected : has a path from each node to every other node and vice versa
    - Weakly connected

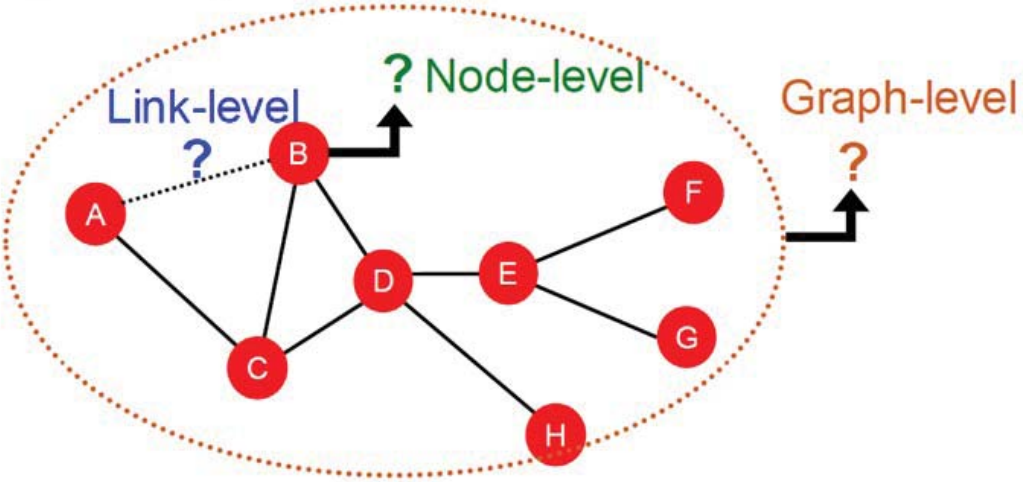


Graph on the left is connected but not strongly connected (e.g., there is no way to get from F to G by following the edge directions).

Traditional Feature-based Methods

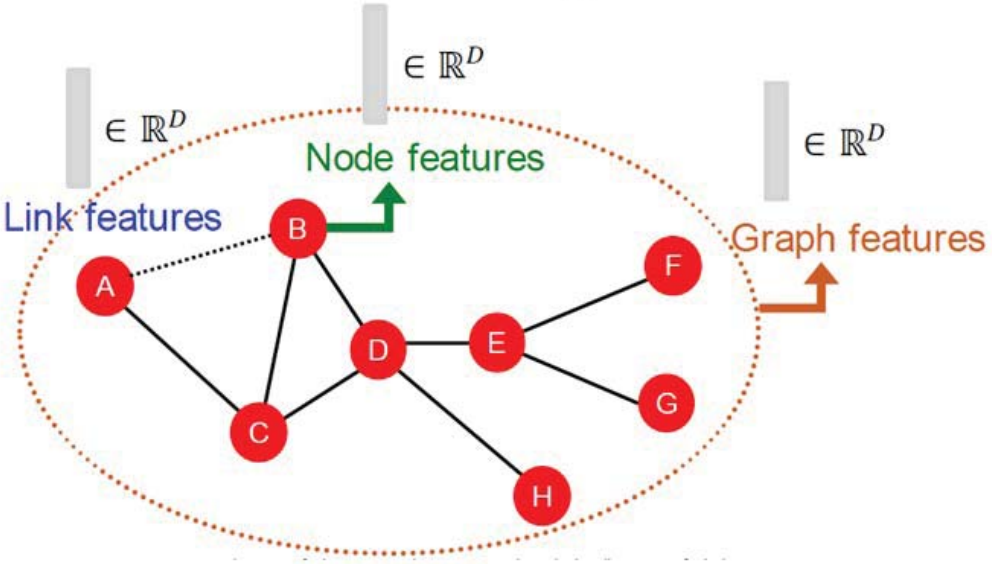
Traditional Feature-based Methods

- Machine Learning Tasks
  - Node-level prediction
  - Link-level prediction
  - Graph-level prediction



Traditional Feature-based Methods

- Machine Learning Pipeline
  - Design features for nodes/links/graphs
  - Obtain features for all training data



Traditional Feature-based Methods

- Machine Learning Pipeline
  - **Train an ML model:**
    - Random forest
    - SVM
    - Neural network, etc.
  - **Apply the model:**
    - Given a new node/link/graph, obtain its features and make a prediction

$$\boldsymbol{x_1} \rightarrow y_1$$
$$\vdots$$
$$\boldsymbol{x_N} \rightarrow y_N$$

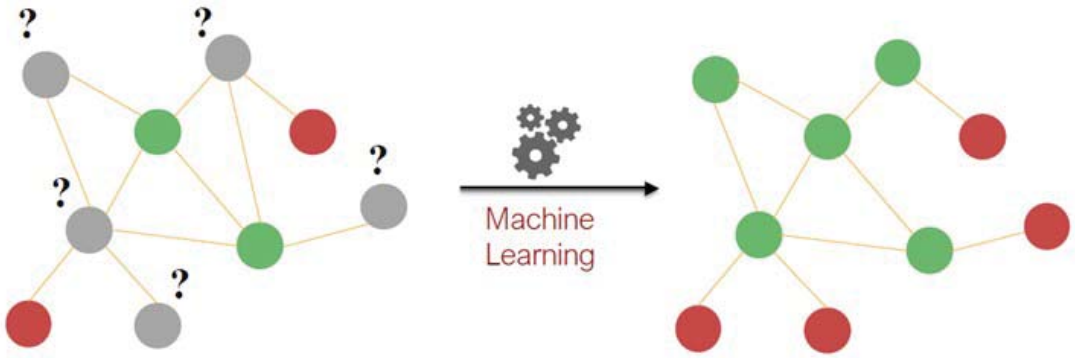
$$\boldsymbol{x} \rightarrow y$$

Traditional Feature-based Methods

- Feature Design
  - Using effective features over graphs is the key to achieve good model performance
  - In this lecture, we overview the traditional features for :
    - Node-level prediction
    - Link-level prediction
    - Graph-level prediction
  - For simplicity, we focus on undirected graphs.

Traditional Feature-based Methods

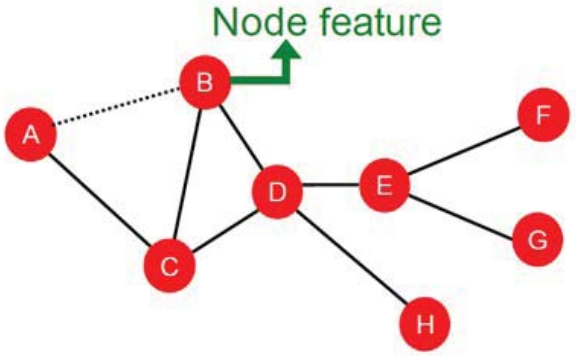
- Node Features
  - Node-Level Tasks



Node classification

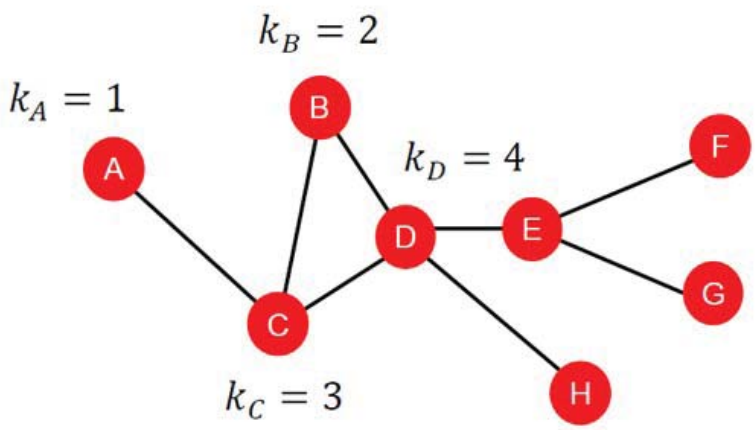
Traditional Feature-based Methods

- Node Features
  - Goal
    - Characterize the structure and position of a node in the network
  - Overview
    - Node degree
    - Node centrality
    - Clustering coefficient
    - Graphlets



Traditional Feature-based Methods

- Node Features : Node degree
  - The degree k_v of node v is the number of edges (neighboring nodes) the node has.
  - Treats all neighboring nodes equally.



Traditional Feature-based Methods

- Node Features : Node centrality
  - Node degree counts the neighboring nodes without capturing their importance.
  - Node centrality c_v takes the node importance in a graph into account
  - **Different ways to model importance:**
    - Eigenvector centrality
    - Betweenness centrality
    - Closeness centrality
    - and many others…

Traditional Feature-based Methods

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

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

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

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

Traditional Feature-based Methods

- Node Features : Node centrality
  - Eigenvector centrality
    - Rewrite the recursive equation in the matrix form.

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

λ is normalization const (largest eigenvalue of A)

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

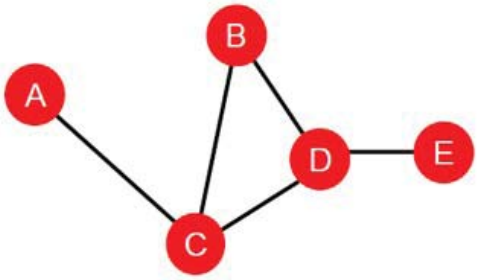
    - We see that centrality c is the **eigenvector of $\boldsymbol{A}$**!
    - The largest eigenvalue λ_{max} is always positive and unique (by Perron-Frobenius Theorem).
    - The eigenvector $\boldsymbol{c}_{max}$ corresponding to λ_{max} is used for centrality.

Traditional Feature-based Methods

- Node Features : Node centrality
  - Betweenness centrality
    - A node is important if it lies on many shortest paths between other nodes.

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

    - **Example:**



$c_A = c_B = c_E = 0$

$c_C = 3$

(A-C-B, A-C-D, A-C-D-E)

$c_D = 3$

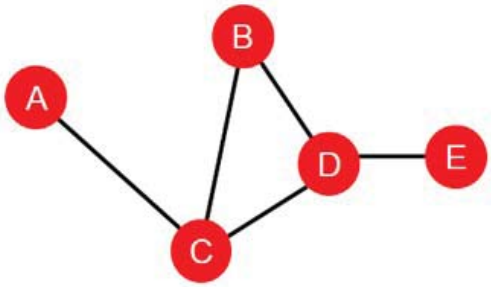
(A-C-D-E, B-D-E, C-D-E)

Traditional Feature-based Methods

- Node Features : Node centrality
  - Closeness centrality
    - A node is important if it has small shortest path lengths to all other nodes.

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

    - **Example:**



$c_A = 1/(2+1+2+3) = 1/8$
(A-C-B, A-C, A-C-D, A-C-D-E)

$c_D = 1/(2+1+1+1) = 1/5$
(D-C-A, D-B, D-C, D-E)

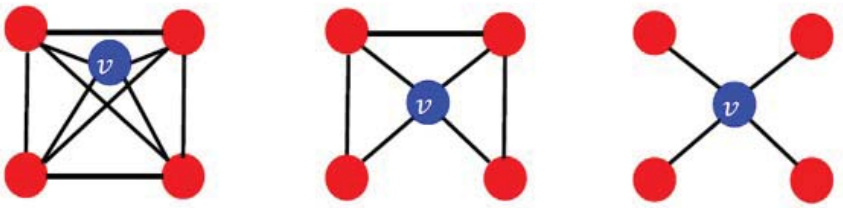
Traditional Feature-based Methods

- 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).

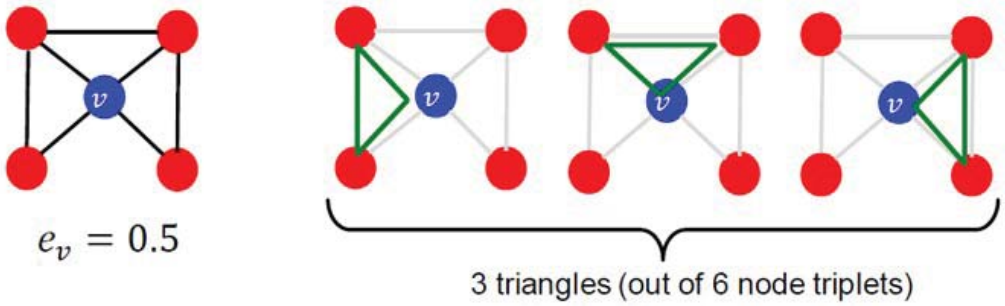
  - **Examples:**



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

Traditional Feature-based Methods

- Node Features : Graphlets
  - **Observation:** Clustering coefficient counts the #(triangles) in the ego-network

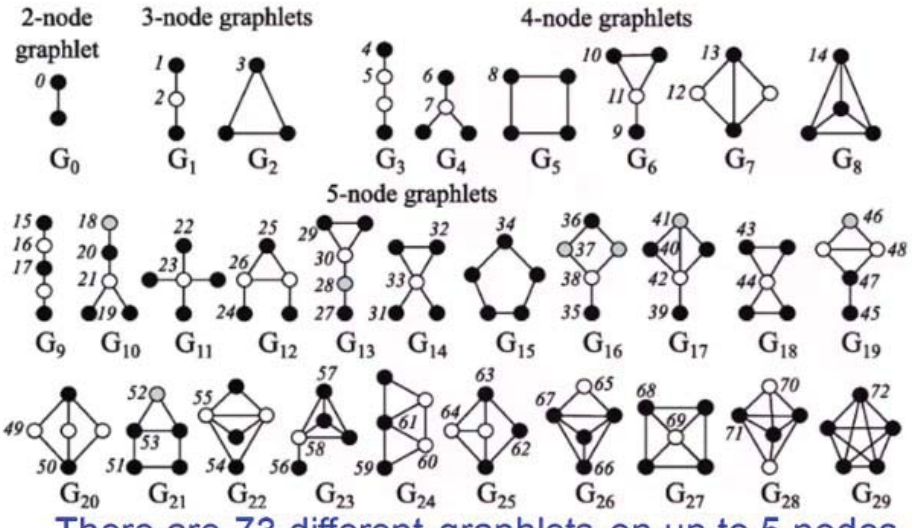


  - We can generalize the above by counting #(pre-specified subgraphs, i.e., graphlets).

Traditional Feature-based Methods

- Node Features : Graphlets

Graphlets: Rooted connected non-isomorphic subgraphs:



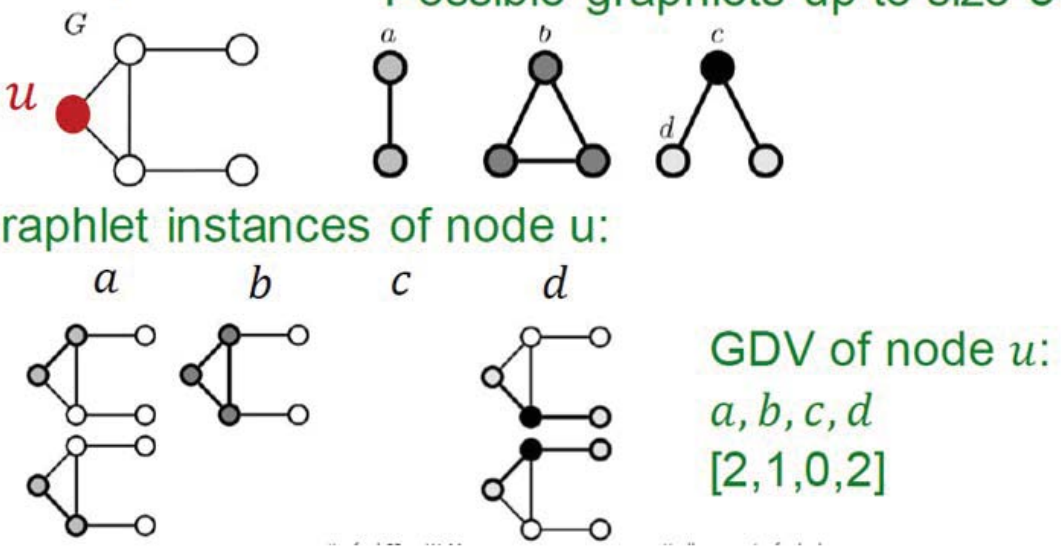
There are 73 different graphlets on up to 5 nodes

Traditional Feature-based Methods

- Node Features : Graphlets
  - **Degree** counts **#(edges)** that a node touches
  - **Clustering coefficient** counts **#(triangles)** that a node touches.
  - **Graphlet Degree Vector (GDV)**: Graphlet-base features for nodes
    - **GDV** counts **#(graphlets)** that a node touches

Traditional Feature-based Methods

- Node Features : Graphlets
  - **Example:**



Traditional Feature-based Methods

- Node Features : Graphlets
  - Considering graphlets on 2 to 5 nodes we get:
    - **Vector of 73 coordinates** is a signature of a node that describes the topology of node's neighborhood
    - Captures its interconnectivities out to a **distance of 4 hops**
  - Graphlet degree vector provides a measure of a **node's local network topology**:
    - Comparing vectors of two nodes provides a more detailed measure of local topological similarity than node degrees or clustering coefficient.

Traditional Feature-based Methods

- Node Features
  - **We have introduced different ways to obtain node features.**
  - **They can be categorized as:**
    - Importance-based features:
      - Node degree
      - Different node centrality measures
    - Structure-based features:
      - Node degree
      - Clustering coefficient
      - Graphlet count vector

Traditional Feature-based Methods

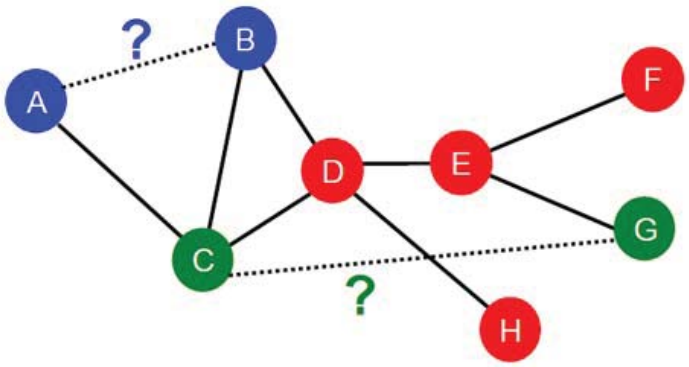
- Node Features
  - **Importance-based features**: capture the importance of a node in a graph
    - Node degree:
      - Simply counts the number of neighboring nodes
    - Node centrality:
      - Models importance of neighboring nodes in a graph
      - Different modeling choices: eigenvector centrality,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

Traditional Feature-based Methods

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

Traditional Feature-based Methods

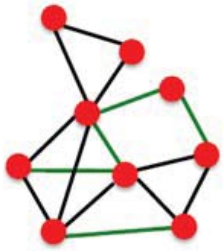
- Link Features
  - The task is to predict **new links** based on the existing links.
  - At test time, node pairs (with no existing links) are ranked, and top K node pairs are predicted.
  - The key is to design features for **a pair of nodes.**



Traditional Feature-based Methods

- Link Features

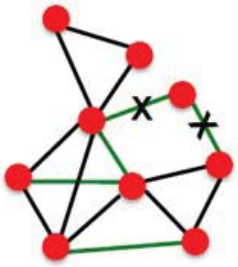
  Two formulations of the link prediction task:
  - **1) Links missing at random:**
    - Remove a random set of links and then aim to predict them
  - **2) Links over time:**
    - Given $G[t_0, t_0']$ a graph defined by edges up to time t_0', **output a ranked list L** of edges (not in $G[t_0, t_0']$) that are predicted to appear in time $G[t_1, t_1']$
    - **Evaluation:**
      - $n = |E_{new}|$: # new edges that appear during the test period $[t_1, t_1']$
      - Take top n elements of L and count correct edges



$G[t_0, t_0']$
$G[t_1, t_1']$

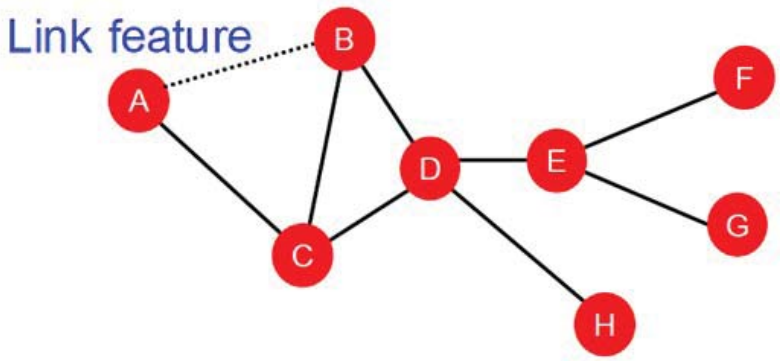
Traditional Feature-based Methods

- Link Features
  - **Methodology:**
    - For each pair of nodes *(x,y)* compute score *c(x,y)*
      - For example, *c(x,y)* could be the # of common neighbors of *x* and *y*
    - Sort pairs *(x,y)* by the decreasing score *c(x,y)*
    - **Predict top n pairs as new links**
    - **See which of these links actually appear in $G[t_1, t_1']$**



Traditional Feature-based Methods

- Link Features
  - Overview
    - Distance-based feature
    - Local neighborhood overlap
    - Global neighborhood overlap

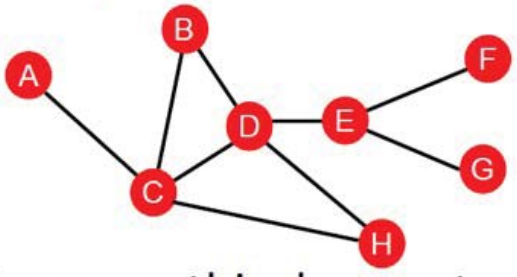


Traditional Feature-based Methods

- Link Features : distance-based feature

Shortest-path distance between two nodes

- Example:



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

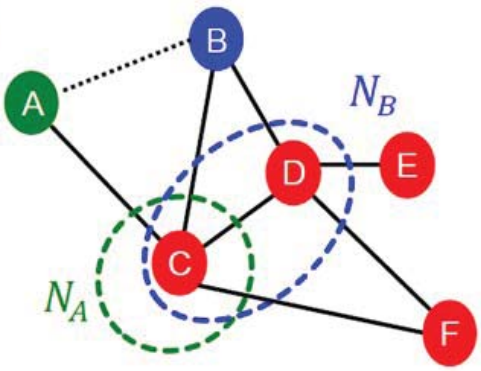
- However, this does not capture the degree of neighborhood overlap:
  - Node pair *(B, H)* has 2 shared neighboring nodes, while pairs *(B, E)* and *(A, B)* only have 1 such node.

Traditional Feature-based Methods

- Link Features : local neighborhood overlap

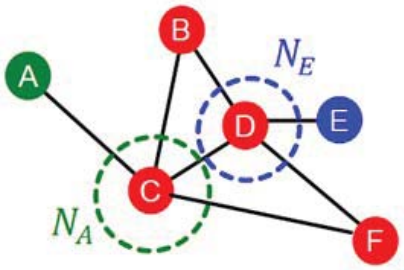
Captures # neighboring nodes shared between two nodes v_1 and v_2:

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



Traditional Feature-based Methods

- Link Features : local neighborhood overlap
  - **Limitation of local neighborhood features:**
    - Metric is always zero if the two nodes do not have any neighbors in common.



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

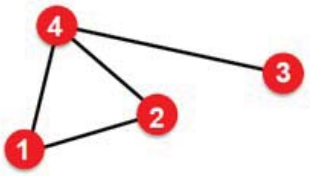
    - However, the two nodes may still potentially be connected in the future.
  - **Global neighborhood overlap** metrics resolve the limitation by considering the entire graph.

Traditional Feature-based Methods

- Link Features : global neighborhood overlap
  - **Katz index:** count the number of walks of all lengths between a pair of nodes.
  - How to compute #walks between two nodes?
  - Use **adjacency matrix powers**!
    - A_{uv} specifies #walks of length 1 (direct neighborhood) between u and v.
    - A_{uv}^2 specifies #walks of **length 2** (neighbor of neighbor) between u and v.
    - And, A_{uv}^l specifies #walks of **length l**.

Traditional Feature-based Methods

- Link Features : global neighborhood overlap
  - Powers of adjacence matrices
    - Computing #walks between two nodes
      - **Recall**: $A_{uv} = 1$ if $u \in N(v)$
      - Let $P_{uv}^{(K)}$ = #walks of length K between u and v
      - We will show $P^{(K)} = A^k$
      - $P_{uv}^{(1)}$ = #walks of length 1 (direct neighborhood) between u and $v = A_{uv}$



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

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

Traditional Feature-based Methods

- Link Features : global neighborhood overlap
  - Powers of adjacence matrices
    - **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

Traditional Feature-based Methods

- Link Features : global neighborhood overlap
  - Katz index between v_1 and v_2 is calculated as

Sum over *all walk lengths*

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

#walks of length l between v_1 and v_2

$0 < \beta < 1$: discount factor

  - Katz index matrix is computed in closed-form:

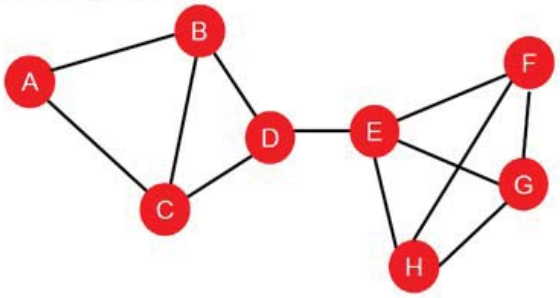
$$\boldsymbol{S} = \sum_{i=1}^{\infty} \beta^i \boldsymbol{A}^i = \underbrace{(\boldsymbol{I} - \beta \boldsymbol{A})^{-1}}_{= \sum_{i=0}^{\infty} \beta^i \boldsymbol{A}^i \text{ by geometric series of matrices}} - \boldsymbol{I},$$

Traditional Feature-based Methods

- Link Features : summary
  - **Distance-based features:**
    - Uses the shortest path length between two nodes but does not capture how neighborhood overlaps.
  - **Local neighborhood overlap:**
    - Captures how many neighboring nodes are shared by two nodes.
    - Becomes zero when no neighbor nodes are shared.
  - **Global neighborhood overlap:**
    - Uses global graph structure to score two nodes.
    - Katz index counts #walks of all lengths between two nodes.

Traditional Feature-based Methods

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



Traditional Feature-based Methods

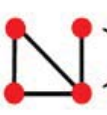
- Graph Features : Kernel Methods
  - **Kernel methods** are widely-used for traditional ML for graph-level prediction.
  - **Idea: Design kernels instead of feature vectors.**
  - **A quick introduction to Kernels:**
    - Kernel $K(G, G') \in \mathbb{R}$ measures similarity b/w data
    - Kernel matrix $\boldsymbol{K} = \left(K(G, G')\right)_{G,G'}$ must always be positive semidefinite (i.e., has positive eigenvalues)
    - There exists a feature representation $\phi(\cdot)$ such that $K(G, G') = \phi(\mathrm{G})^{\mathrm{T}}\phi(G')$
    - Once the kernel is defined, off-the-shelf ML model, such as kernel SVM, can be used to make predictions.

Traditional Feature-based Methods

- Graph Features : Graph Kernel
  - **Graph Kernels:** Measure similarity between two graphs:
    - Graphlet Kernel [1]
    - Weisfeiler-Lehman Kernel [2]
    - Other kernels are also proposed in the literature (beyond the scope of this lecture)
      - Random-walk kernel
      - Shortest-path graph kernel
      - And many more…

Traditional Feature-based Methods

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

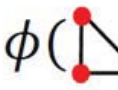
$\phi($  $) = \phi($  $)$

Traditional Feature-based Methods

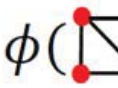
- Graph Features : Graph Kernel

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

  Deg1: ● Deg2: ● Deg3: ●

  ϕ() = count() = [1, 2, 1]

  Obtains different features for different graphs!

  ϕ() = count() = [0, 2, 2]

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

Traditional Feature-based Methods

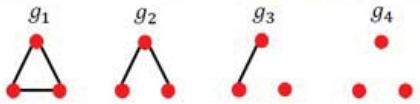
- Graph Features : Graphlet Features
  - **Key idea:** Count the number of different graphlets in a graph.
    - **Note:** Definition of graphlets here is slightly different from node-level features.
    - The two differences are:
      - Nodes in graphlets here do **not need to be connected** (allows for isolated nodes)
      - The graphlets here are not rooted.

Traditional Feature-based Methods

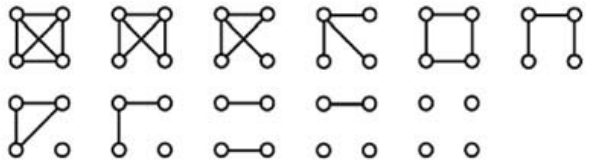
Graph Features : Graphlet Features

Let $\mathcal{G}_k = (g_1, g_2, \ldots, g_{n_k})$ be a list of graphlets of size k.

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



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

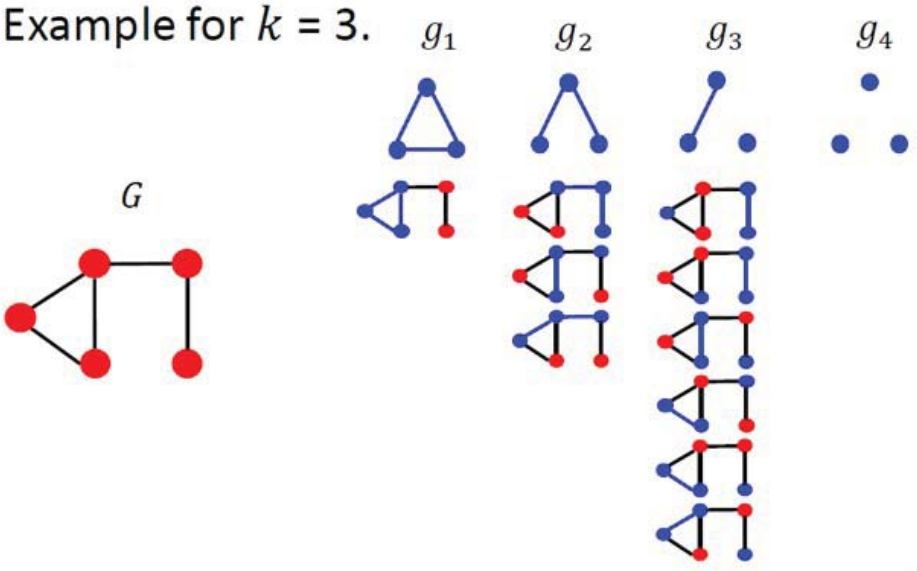


- Given graph G, and a graphlet list $\mathcal{G}_k = (g_1, g_2, \ldots, g_{n_k})$, define the graphlet count vector $\boldsymbol{f}_G \in \mathbb{R}^{n_k}$ as $(\boldsymbol{f}_G)_i = \#(g_i \subseteq G)$ for $i = 1, 2, \ldots, n_k$.

Traditional Feature-based Methods

Graph Features : Graphlet Features

- Example for $k = 3$.



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

Traditional Feature-based Methods

Graph Features : Graphlet Kernel

- Given two graphs, G and G', graphlet kernel is computed as

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

- **Problem:** if G and G' have different sizes, that will greatly skew the value.
- **Solution:** normalize each feature vector

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

Limitations: Counting graphlets is expensive!

- Counting size-k graphlets for a graph with size n by enumeration takes n^k.

Traditional Feature-based Methods

Graph Features : Weisfeiler-Lehman Kernel

- **Goal**: Design an efficient graph feature descriptor $\phi(G)$
- **Idea:** Use neighborhood structure to iteratively enrich node vocabulary.
  - Generalized version of **Bag of node degrees** since node degrees are one-hop neighborhood information.
- **Algorithm to achieve this**:

Color refinement

Traditional Feature-based Methods

- Graph Features : Weisfeiler-Lehman Kernel
  - Color Refinement
    - **Given:** A graph G with a set of nodes V.
      - Assign an initial color $c^{(0)}(v)$ to each node v.
      - Iteratively refine node colors by

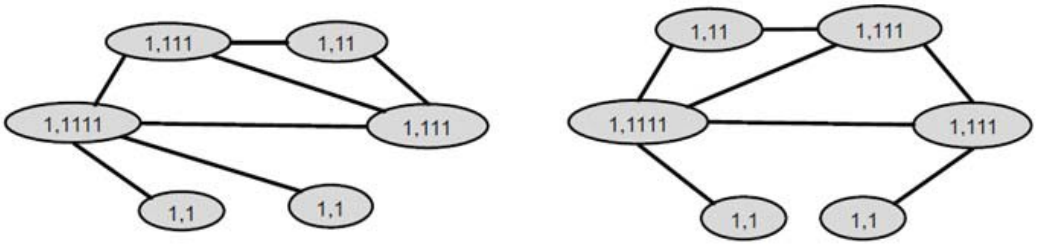
$$c^{(k+1)}(v) = \text{HASH}\left(\left\{c^{(k)}(v), \{c^{(k)}(u)\}_{u \in N(v)}\right\}\right),$$

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

Traditional Feature-based Methods

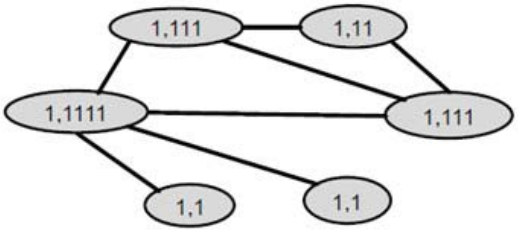
- Graph Features : Weisfeiler-Lehman Kernel
  - Color Refinement (example 1)
    - Assign initial colors

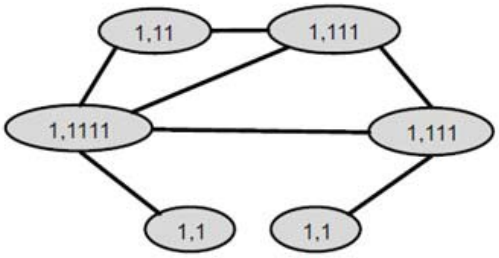
    - Aggregate neighboring colors



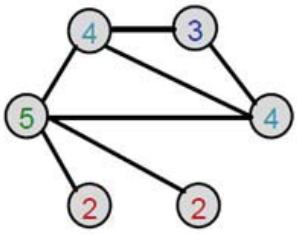
Traditional Feature-based Methods

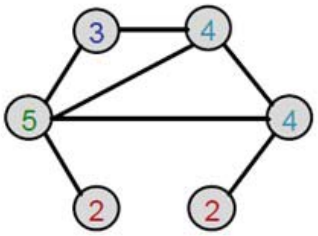
- Graph Features : Weisfeiler-Lehman Kernel
  - Color Refinement (example 2)
    - Aggregated colors





    - Hash aggregated colors



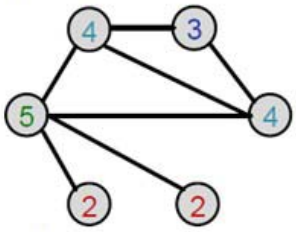


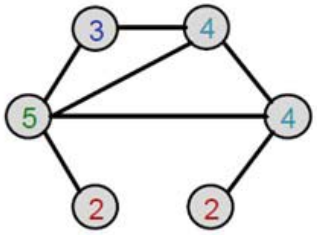
Hash table

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

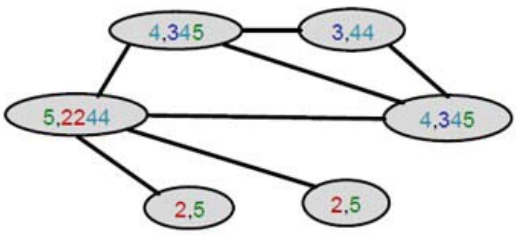
Traditional Feature-based Methods

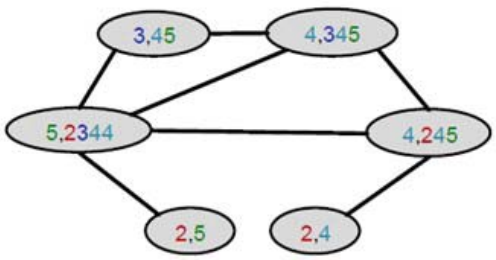
- Graph Features : Weisfeiler-Lehman Kernel
  - Color Refinement (example 3)
    - Aggregated colors





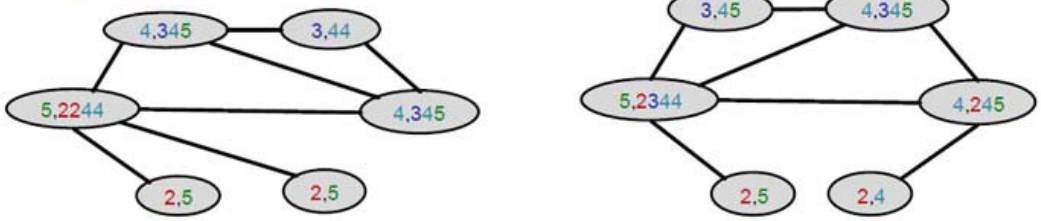
    - Hash aggregated colors



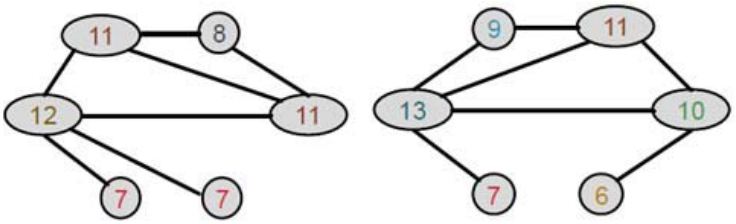


Traditional Feature-based Methods

- Graph Features : Weisfeiler-Lehman Kernel
  - Color Refinement (example 4)
    - Aggregated colors



    - Hash aggregated colors



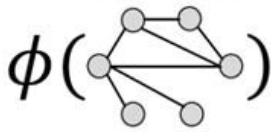
Hash table

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

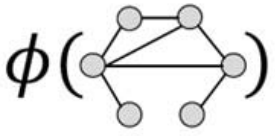
Traditional Feature-based Methods

- Graph Features : Weisfeiler-Lehman Kernel

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

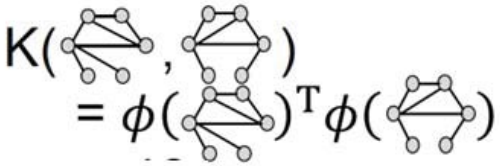


Colors
1,2,3,4,5,6,7,8,9,10,11,12,13
= [6,2,1,2,1,0,2,1,0, 0, 2,1,0]
Counts



1,2,3,4,5,6,7,8,9,10,11,12,13
= [6,2,1,2,1,1,1,0,1, 1, 1, 0, 1]

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



Traditional Feature-based Methods

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

Traditional Feature-based Methods

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