DSC250: Advanced Data Mining

Graph Mining

Zhiting Hu

Lecture 10, Feb 6, 2025

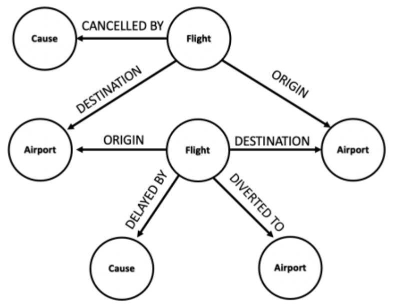
UC San Diego
HALICIOĞLU DATA SCIENCE INSTITUTE

Outline

- Graph features
- Graph representation learning

- Presentation
    - Lila Horwitz: “expainable AI and LIME”

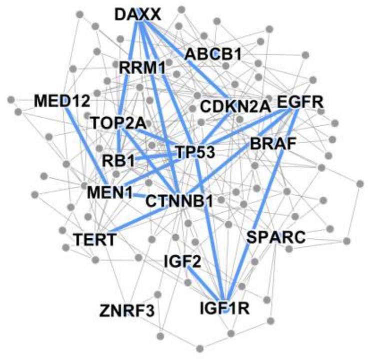
Graph is everywhere



Event Graphs

Image credit: SalientNetworks

Computer Networks



Disease Pathways

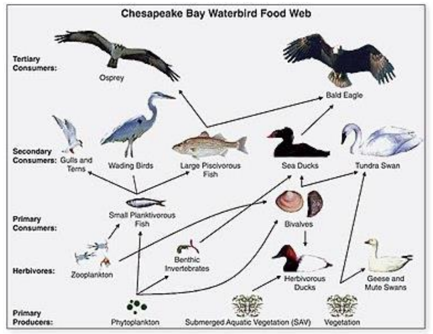


Image credit: Wikipedia

Food Webs

Image credit: Pinterest

Particle Networks

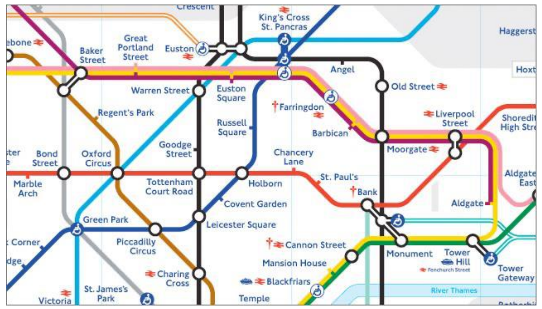


Image credit: visitlondon.com

Underground Networks

Graph is everywhere

Image credit: Medium

Social Networks

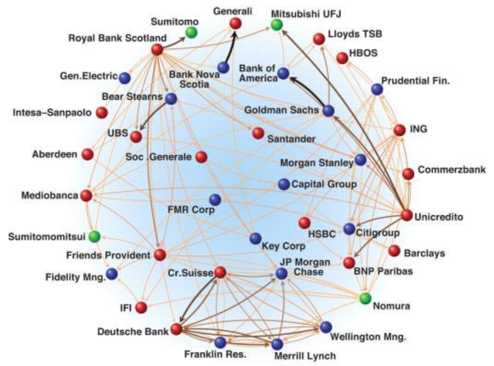
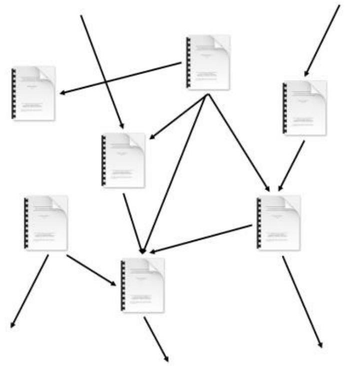


Image credit: Science

Economic Networks

Image credit: Lumen Learning

Communication Networks



Citation Networks

Image credit: Missoula Current News

Internet

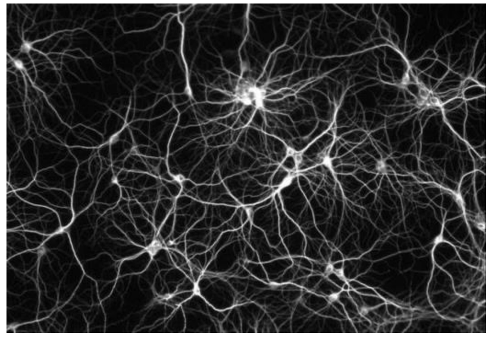


Image credit: The Conversation

Networks of Neurons

Graph is everywhere

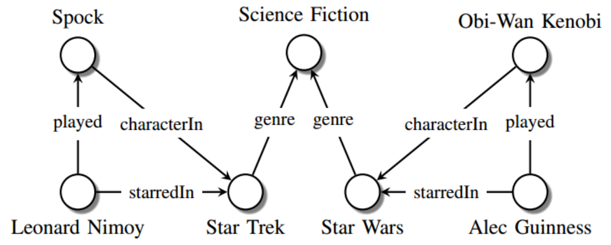


Image credit: Maximilian Nickel et al

Knowledge Graphs

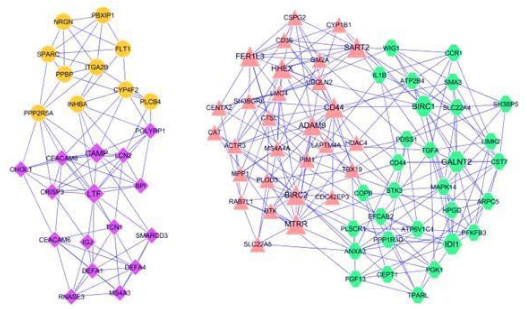


Image credit: ese.wustl.edu

Regulatory Networks

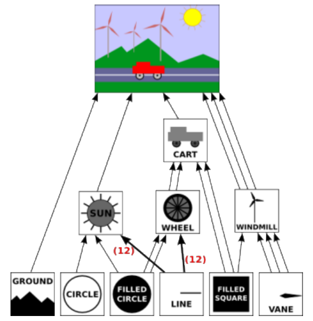


Image credit: math.hws.edu

Scene Graphs

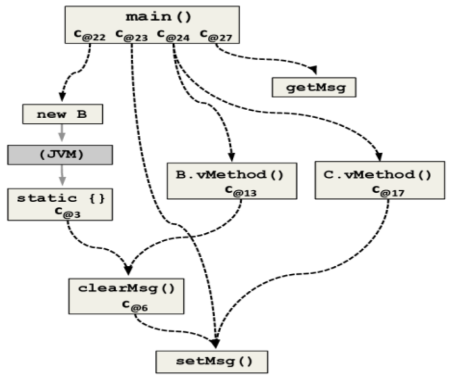


Image credit: ResearchGate

Code Graphs

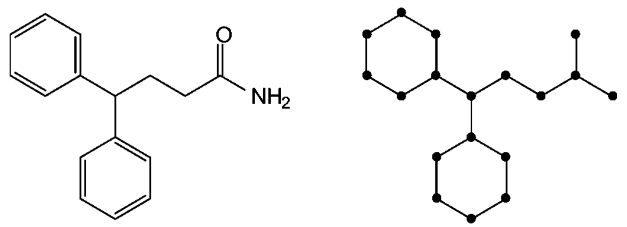


Image credit: MDPI

Molecules

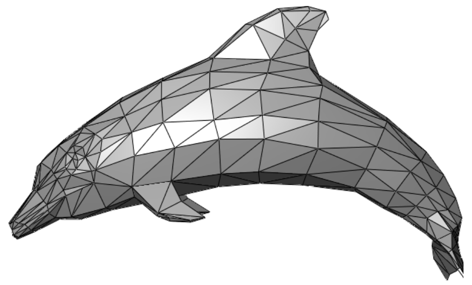
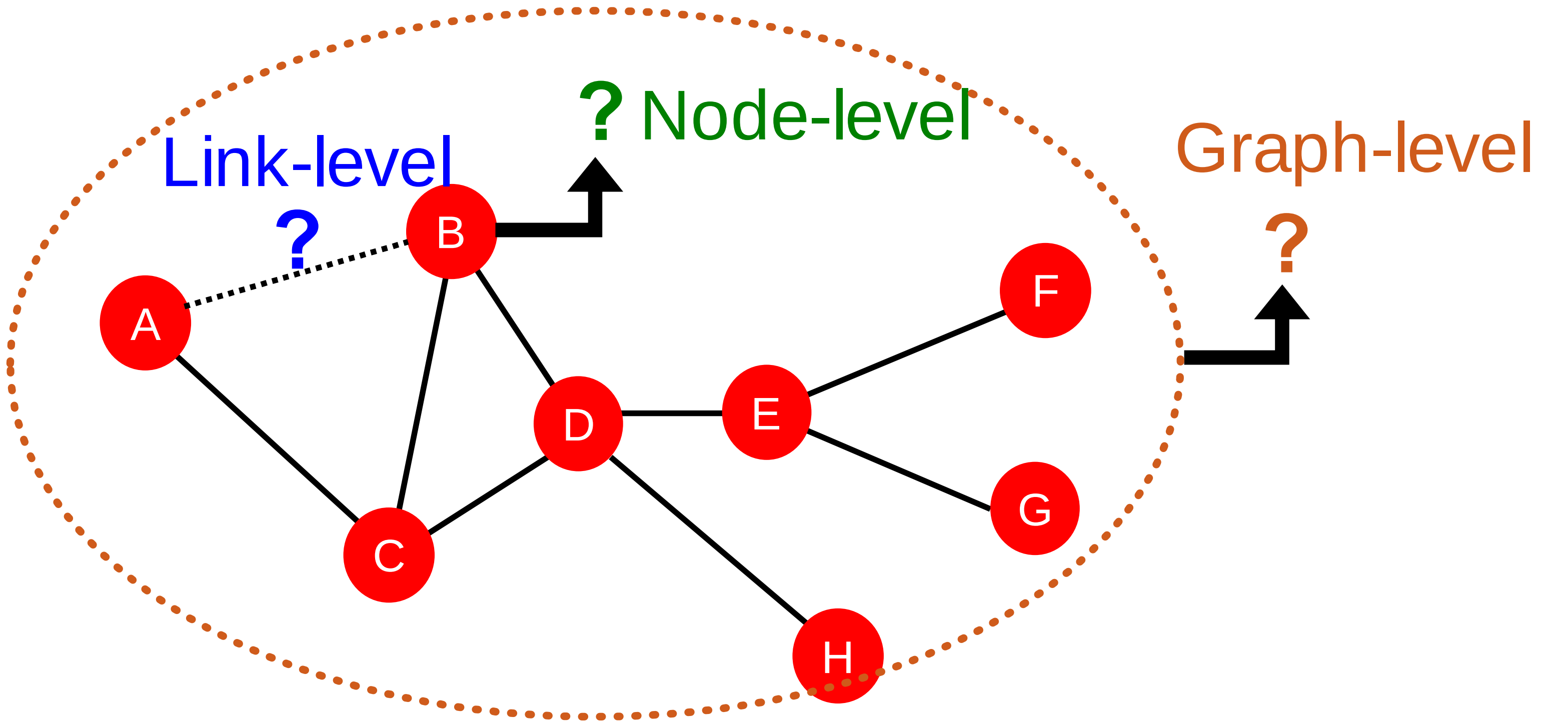


Image credit: Wikipedia

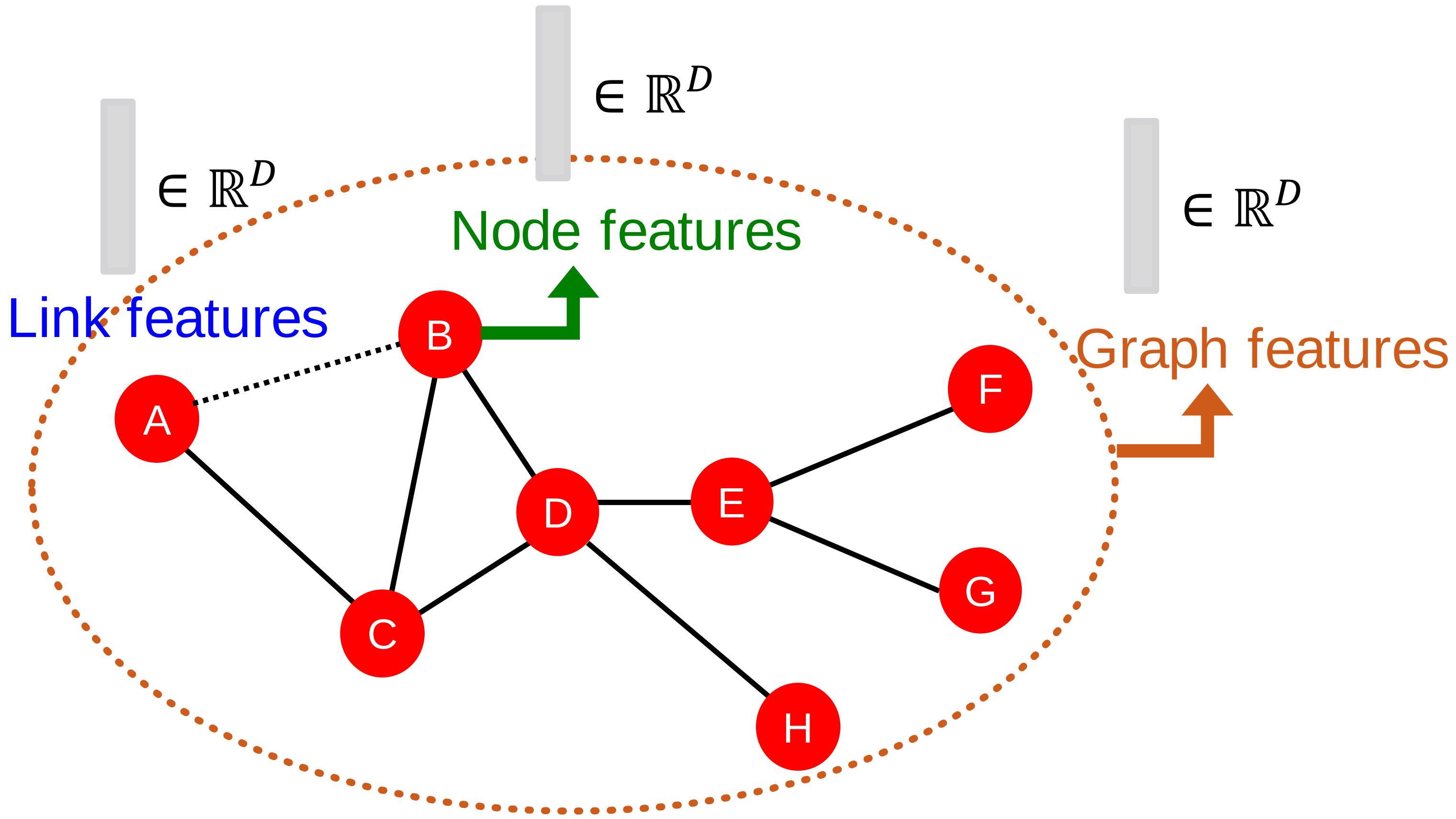
3D Shapes

Tasks on Graph

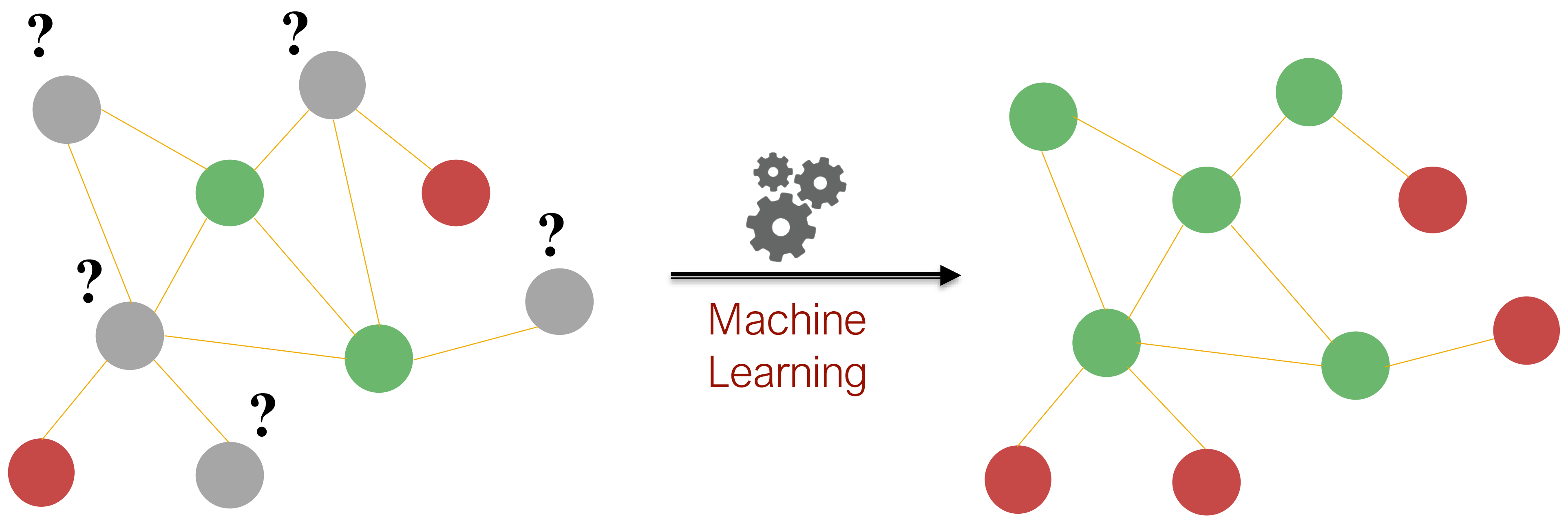
- Node-level prediction
- Link-level prediction
- Graph-level prediction



Getting Features for Nodes/Links/Graphs



Node-level Tasks

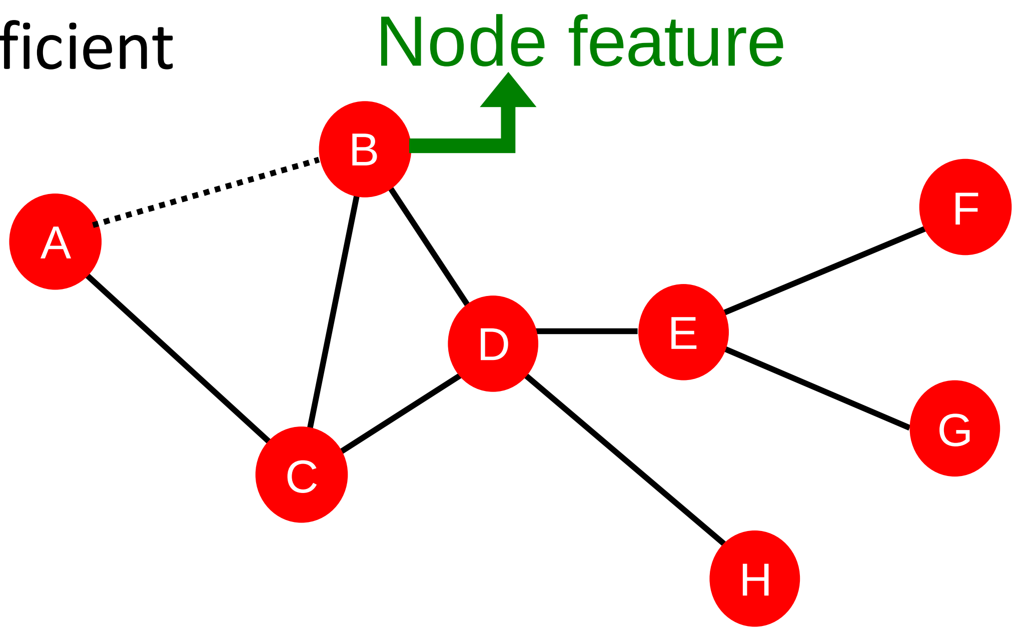


Node classification

Node-level Features

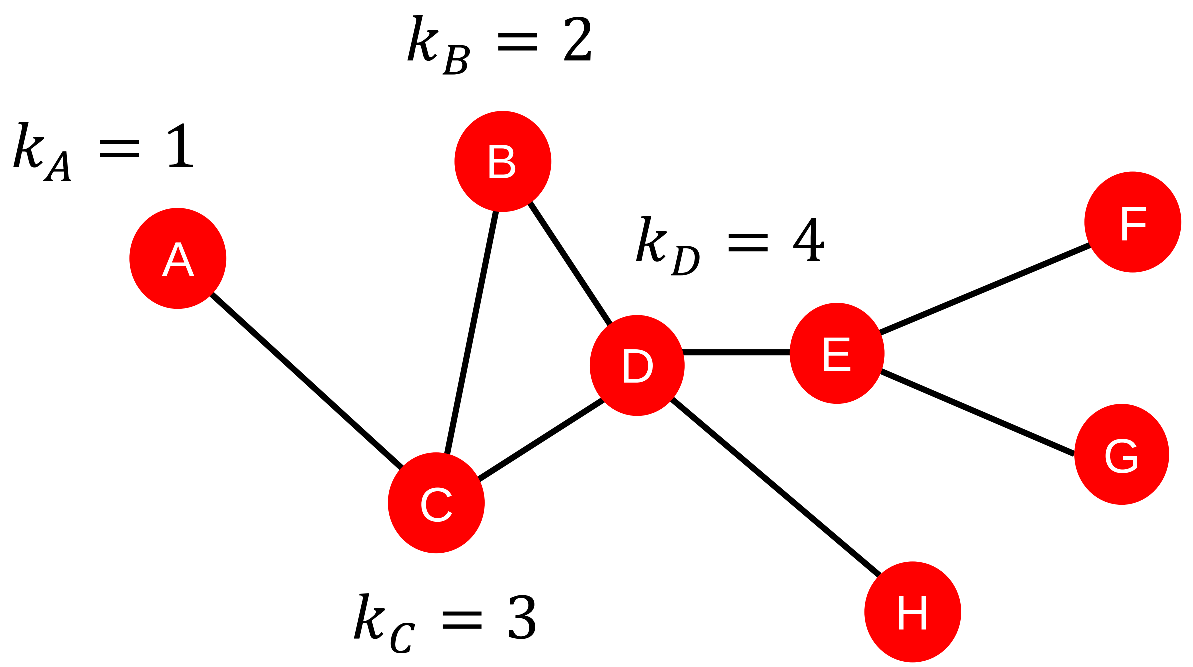
Goal: Characterize the structure and position of a node in the network:

- Node degree
- Node centrality
- Clustering coefficient
- Graphlets



Node-level Features (1): Node Degree

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



Node-level Features (2): 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…

Node-level Features (2): 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)

Node-level Features (2): 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?**

Node-level Features (2): 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

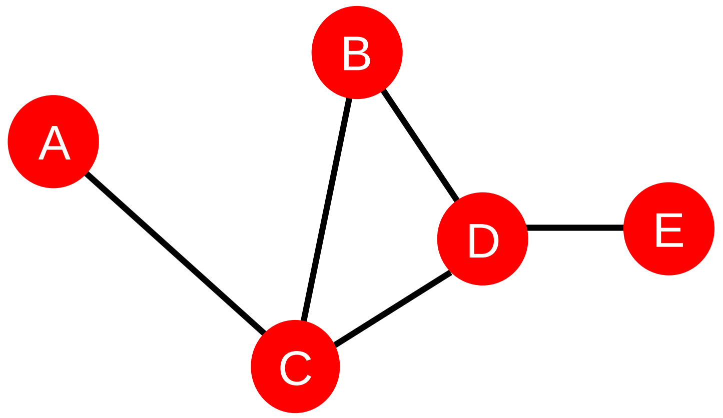
  - We see that centrality $\boldsymbol{c}$ is the **eigenvector of $\boldsymbol{A}$**!
  - The largest eigenvalue λ_{max} is always positive and unique (by Perron-Frobenius Theorem).
  - The eigenvector $\boldsymbol{c}_{max}$ corresponding to λ_{max} is used for centrality.

Node-level Features (2): 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-<u>C</u>-B, A-<u>C</u>-D, A-<u>C</u>-D-E)

$c_D = 3$

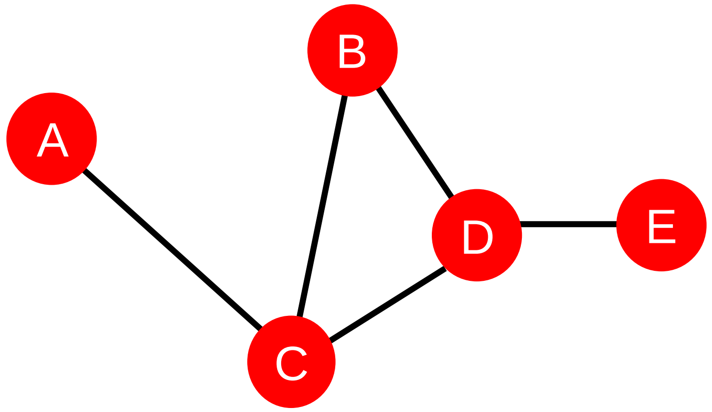
(A-C-<u>D</u>-E, B-<u>D</u>-E, C-<u>D</u>-E)

Node-level Features (2): 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)

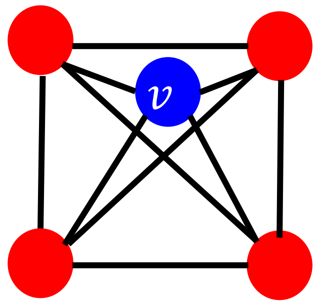
Node-level Features (3): Clustering Coefficient

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

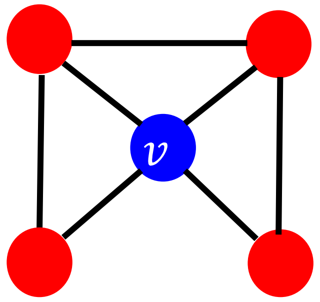
$$e_v = \frac{\#(\text{edges among neighboring nodes})}{\boxed{\binom{k_v}{2}}} \in [0,1]$$

#(node pairs among k_v neighboring nodes)
In our examples below the denominator is 6 (4 choose 2).

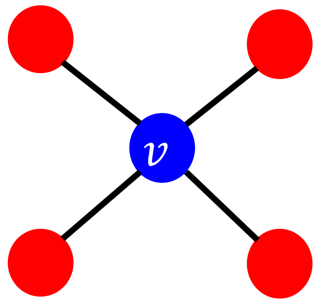
- **Examples:**



$e_v = 1$



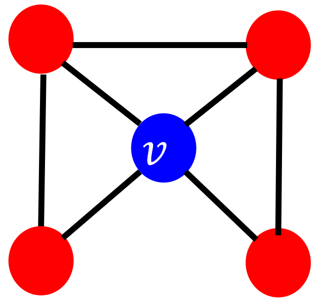
$e_v = 0.5$



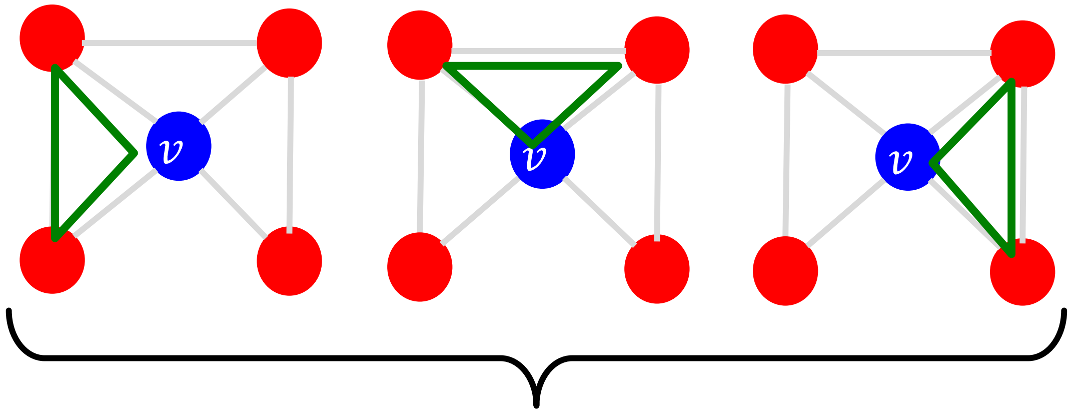
$e_v = 0$

Node-level Features (4): Graphlets

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



$e_v = 0.5$

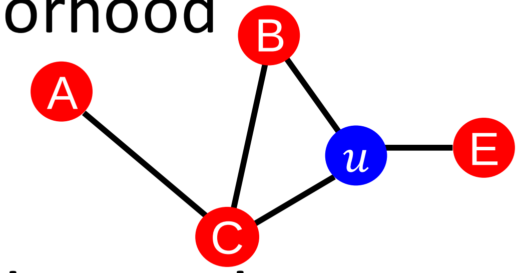


3 triangles (out of 6 node triplets)

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

Node-level Features (4): Graphlets

- **Goal:** Describe network structure around node u
  - **Graphlets** are small subgraphs that describe the structure of node u's network neighborhood

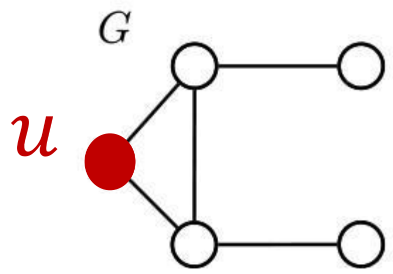


Analogy:

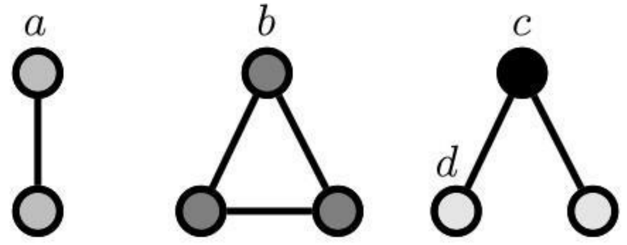
- **Degree** counts **#(edges)** that a node touches
- **Clustering coefficient** counts **#(triangles)** that a node touches.
- **Graphlet Degree Vector (GDV):** Graphlet-base features for nodes
  - **GDV** counts **#(graphlets)** that a node touches

Node-level Features (4): Graphlets

- **Graphlet Degree Vector (GDV):** A count vector of graphlets rooted at a given node.
- **Example:**

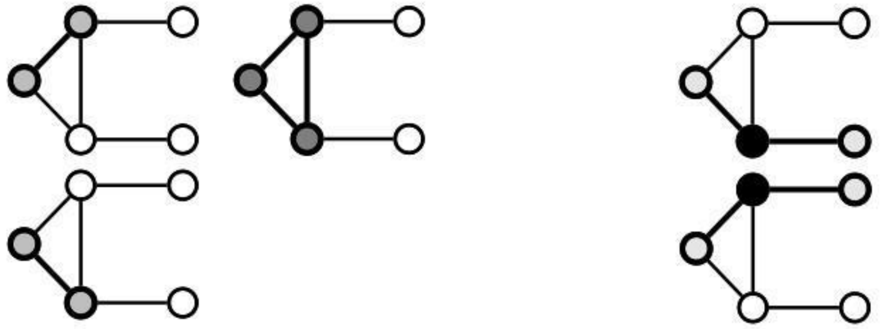


Possible graphlets on up to 3 nodes



Graphlet instances of node u:

a b c d



GDV of node u:

a, b, c, d

[2,1,0,2]

Node-level Features: Summary

- **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

Node-level Features: Summary

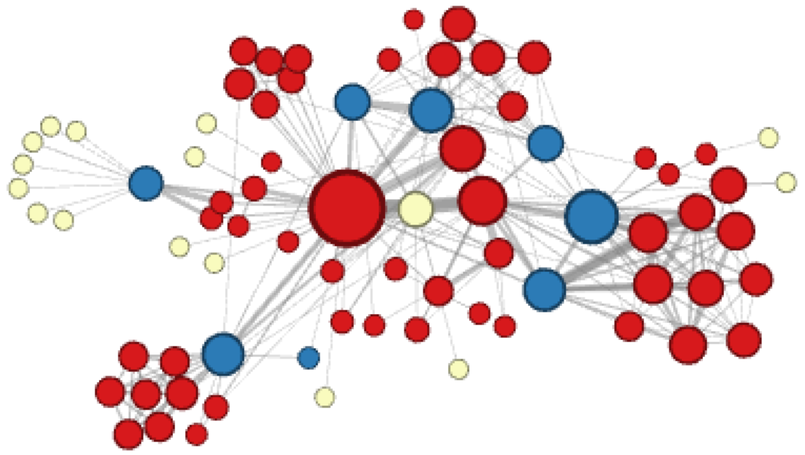
- **Importance-based features**: capture the importance of a node in a graph
  - Node degree:
    - Simply counts the number of neighboring nodes
  - Node centrality:
    - Models importance of neighboring nodes in a graph
    - Different modeling choices: eigenvector centrality, betweenness centrality,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

Node-level Features: Summary

- **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.

Node-level Features: Discussion

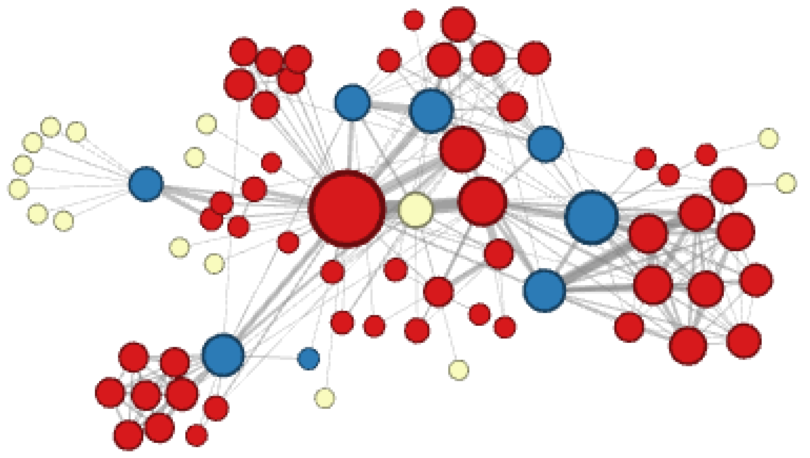
Different ways to label nodes of the network:

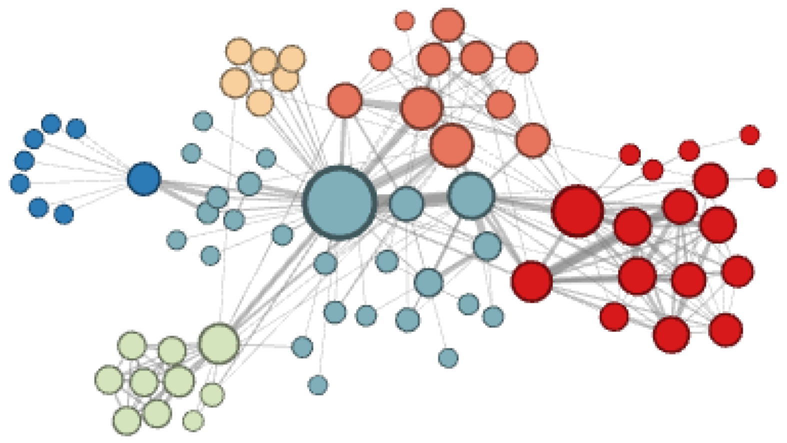


Node features defined so far would allow to distinguish nodes in the above example

Node-level Features: Discussion

Different ways to label nodes of the network:



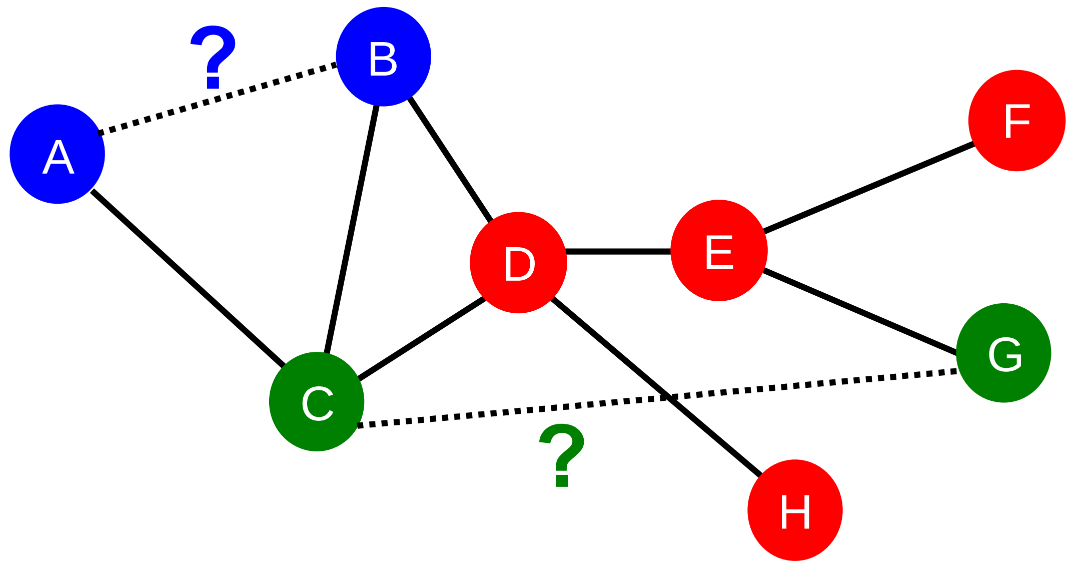


Node features defined so far would allow to distinguish nodes in the above example

?

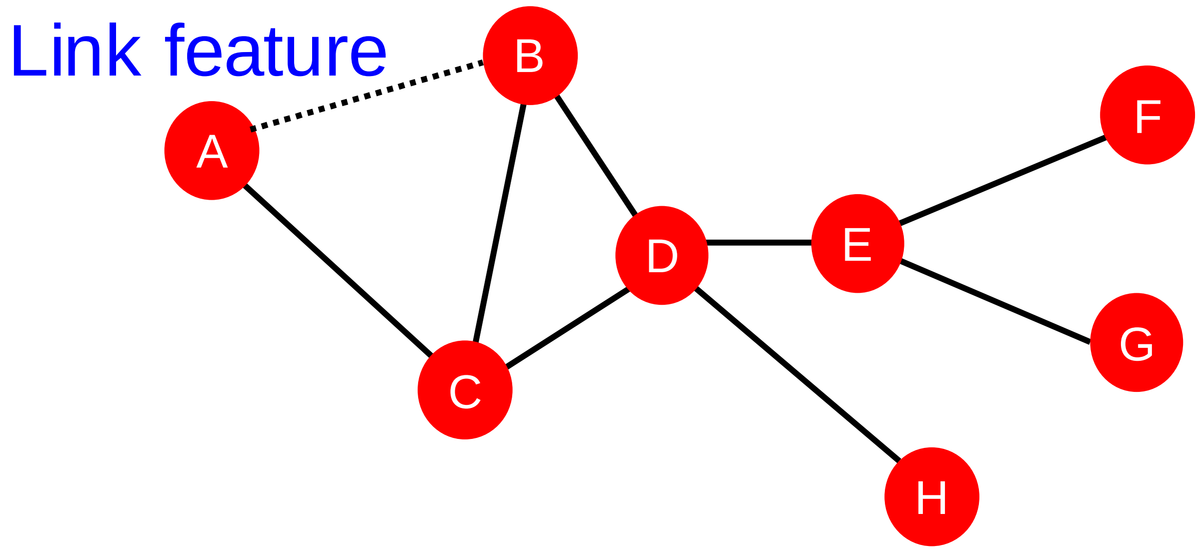
Link-level Task

- 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**.



Link-level Features: Quick Overview

- Distance-based feature
- Local neighborhood overlap
- Global neighborhood overlap



Link-level Features: Quick Overview

- **Distance-based features:**
  - Uses the shortest path length between two nodes but does not capture how neighborhood overlaps.

Link-level Features: Quick Overview

- **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.

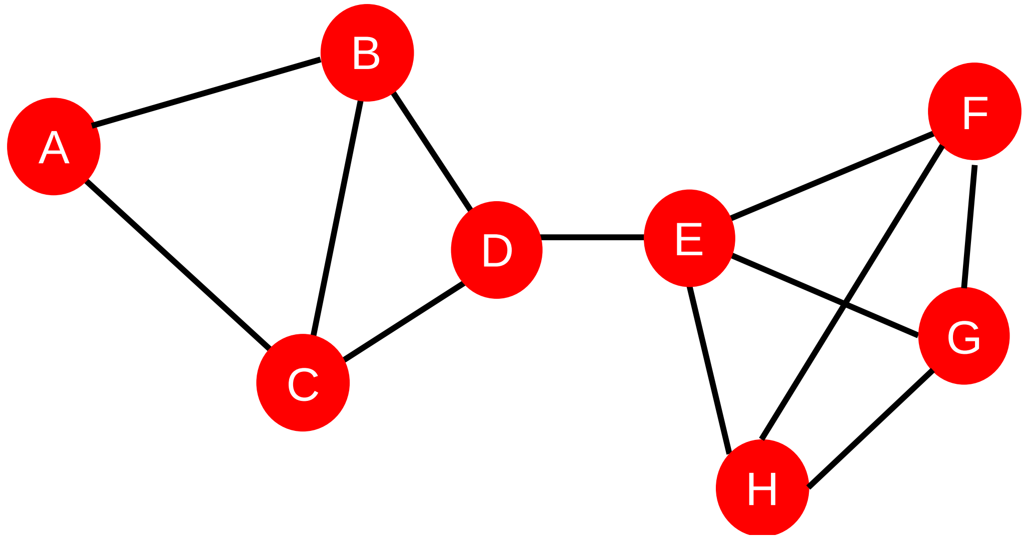
Link-level Features: Quick Overview

- **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.

Graph-level Features

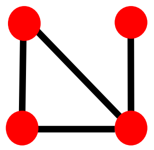
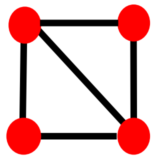
- **Goal:** We want features that characterize the structure of an entire graph.

- **For example:**



Graph-level Features

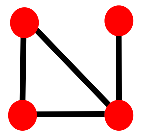
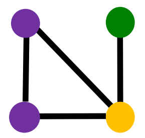
- **Key idea**: Bag-of-Words (BoW) for a graph
  - **Recall:** BoW simply uses the word counts as features for documents (no ordering considered).
  - Naïve extension to a graph: Regard nodes as words.
  - Since both graphs have 4 red nodes, we get the same feature vector for two different graphs…

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

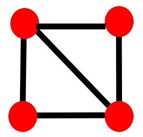
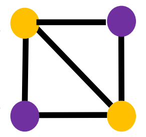
Graph-level Features

What if we use Bag of node degrees?

Deg1: ● Deg2: ● Deg3: ●

$\phi($  $)$ = count() = [1, 2, 1]

$\neq$ Obtains different features for different graphs!

$\phi($  $)$ = 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!

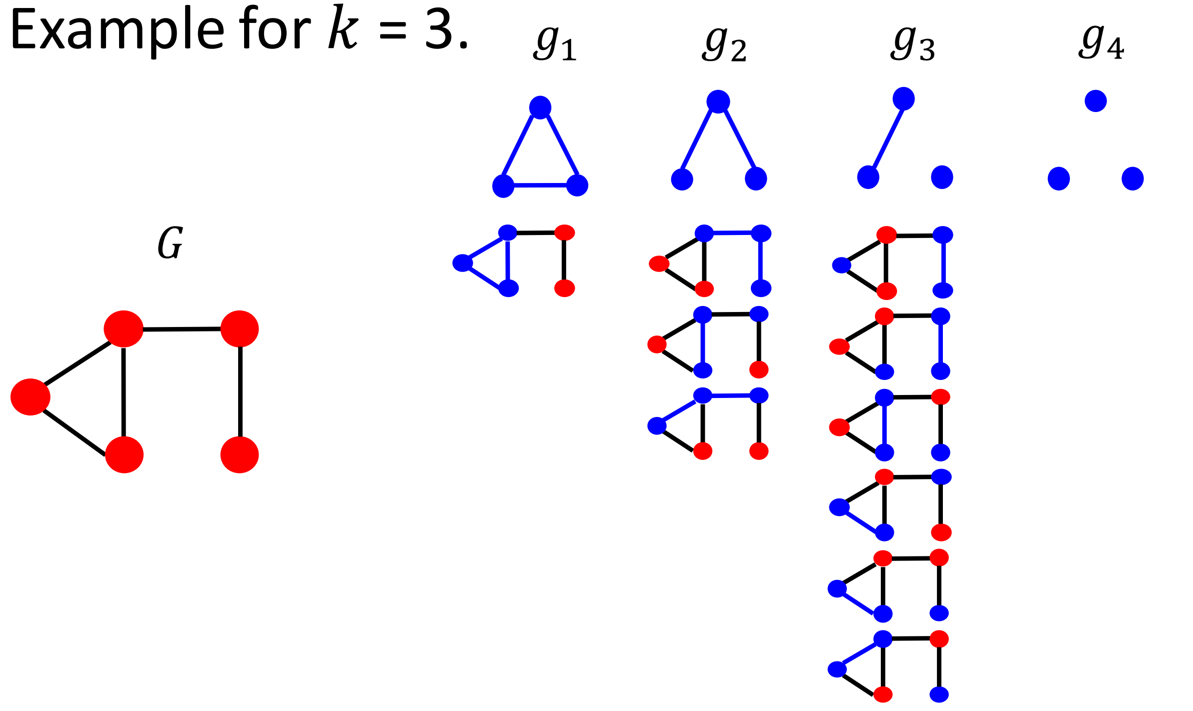
Graph-level Features: Graphlet Features

- **Key idea:** Count the number of **different graphlets** in a graph.
- Given graph G, and a graphlet list $\mathcal{G}_k = (g_1, g_2, \ldots, g_{n_k})$, define the graphlet count vector $\boldsymbol{f}_G \in \mathbb{R}^{n_k}$ as

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

Graph-level Features: Graphlet Features

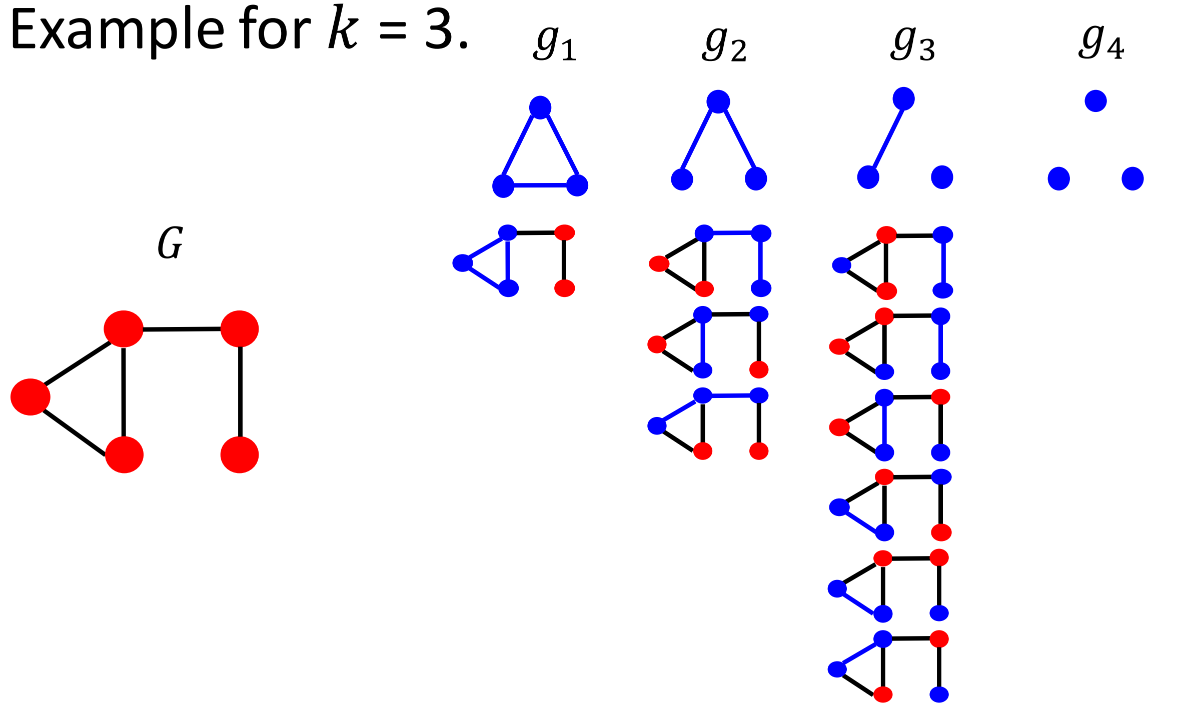
- Example for k = 3.



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

Graph-level Features: Graphlet Features

- Example for k = 3.

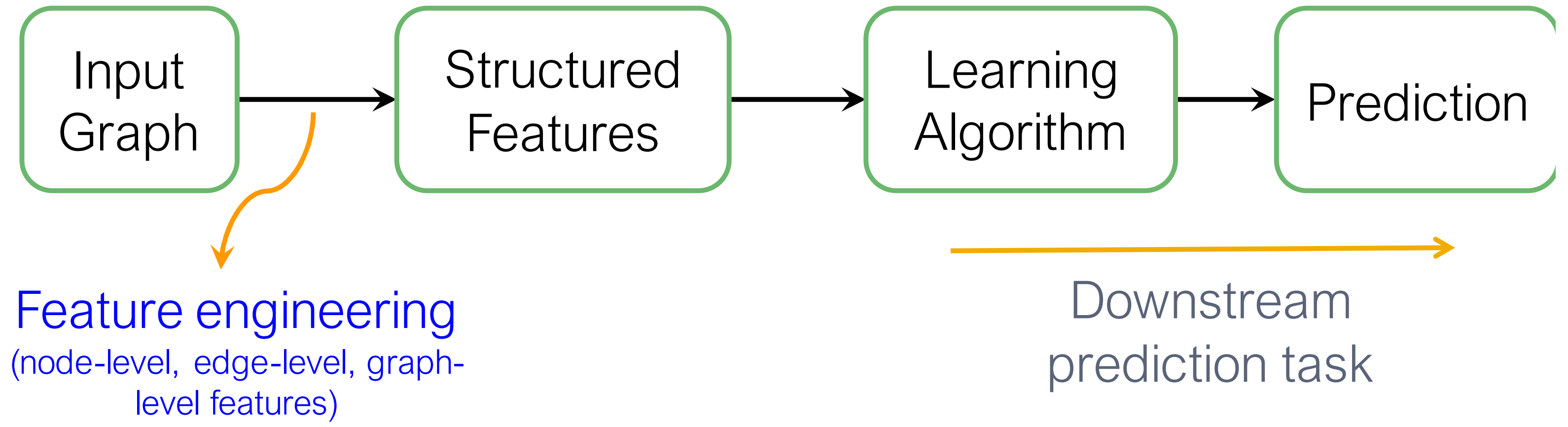


$f_G = (1, \quad 3, \quad 6, \quad 0)^{\mathrm{T}}$

- **Limitations:** Counting graphlets is expensive!
- More advanced methods: color refinement, etc.

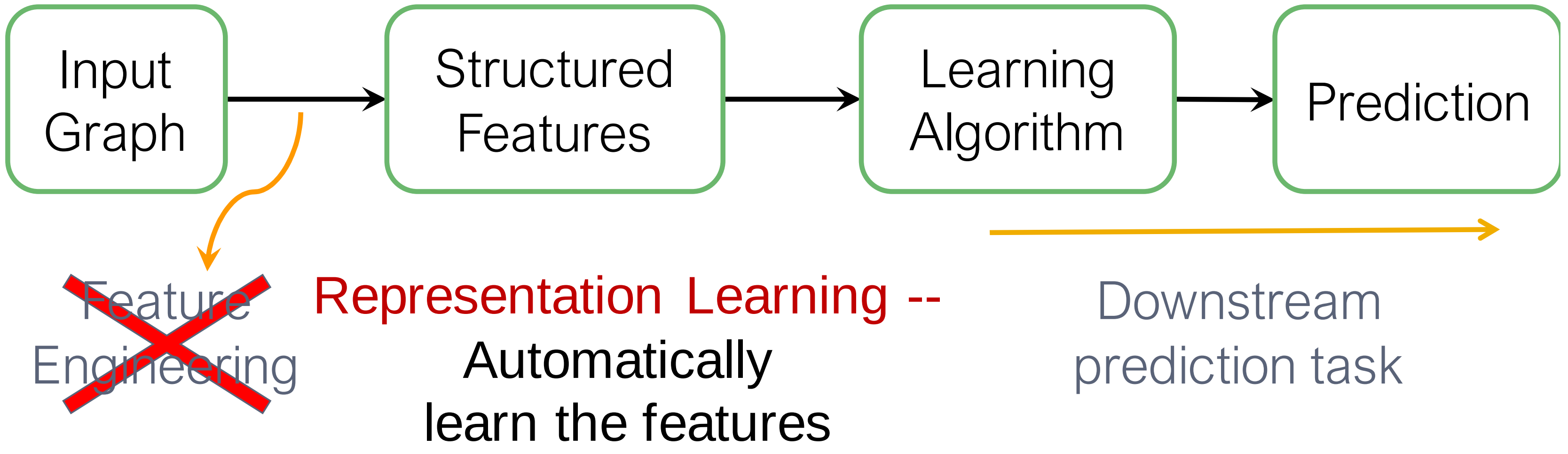
Summary so far: feature engineering

- Node-level:
  - Node degree, centrality, clustering coefficient, graphlets
- Link-level:
  - Distance-based feature
  - Local/global neighborhood overlap
- Graph-level:
  - Graphlet kernel



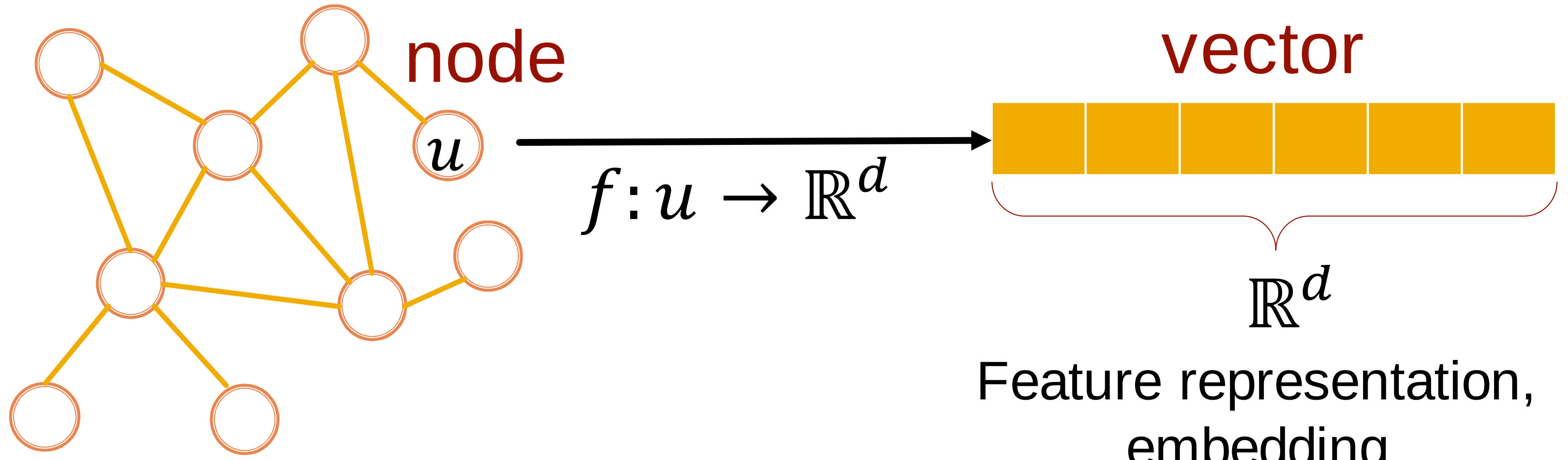
Graph Representation Learning

Graph Representation Learning alleviates the need to do feature engineering every single time.


Input Graph
Structured Features
Learning Algorithm
Prediction
~~Feature Engineering~~
Representation Learning --
Automatically learn the features
Downstream prediction task

Graph Representation Learning

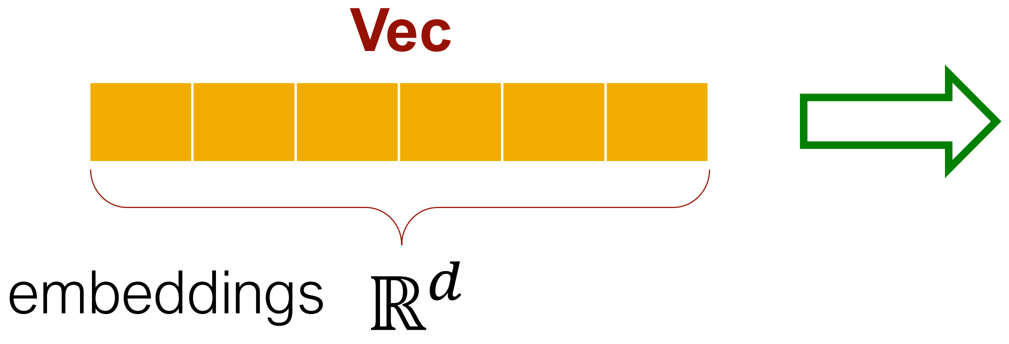
Goal: Efficient task-independent feature learning for machine learning with graphs!



Feature representation, embedding

Node Embedding

- **Task: Map nodes into an embedding space**
  - Similarity of embeddings between nodes indicates their similarity in the network. For example:
    - Both nodes are close to each other (connected by an edge)
  - Encode network information
  - Potentially used for many downstream predictions

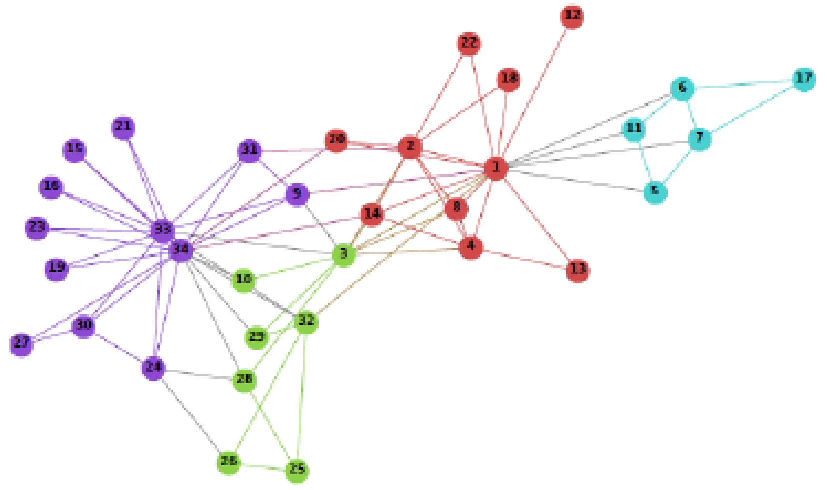


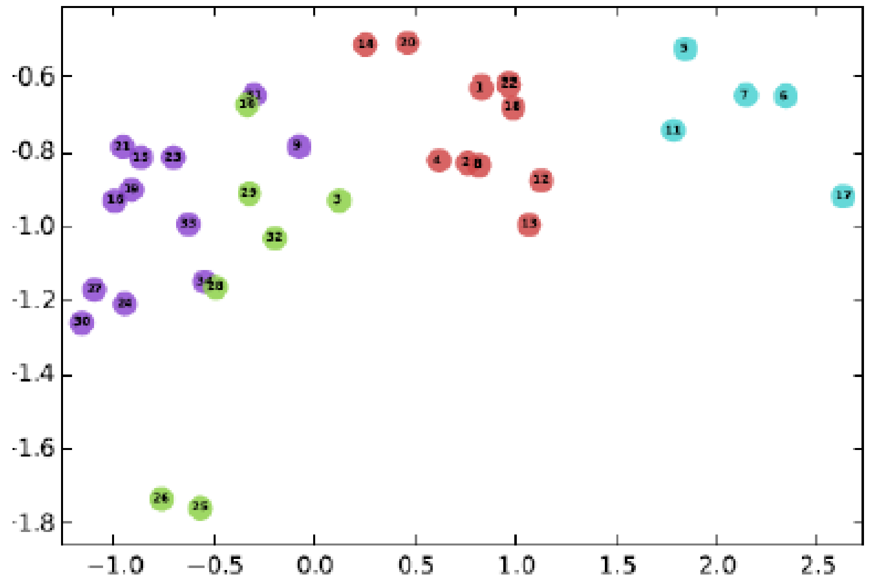
Tasks

- Node classification
- Link prediction
- Graph classification
- Anomalous node detection
- Clustering
- ….

Example Node Embedding

- **2D embedding of nodes of the Zachary's Karate Club network:**

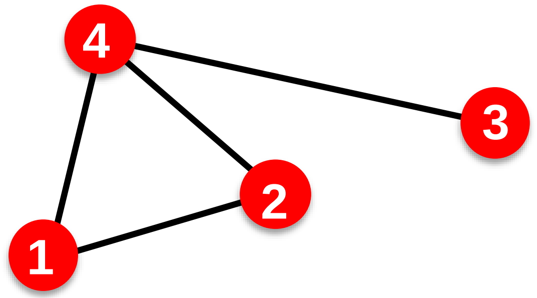

Input



Output

Node Embedding: Setup

- **Assume we have a graph G:**
  - V is the vertex set.
  - **A** is the adjacency matrix (assume binary).
  - **For simplicity: No node features or extra information is used**

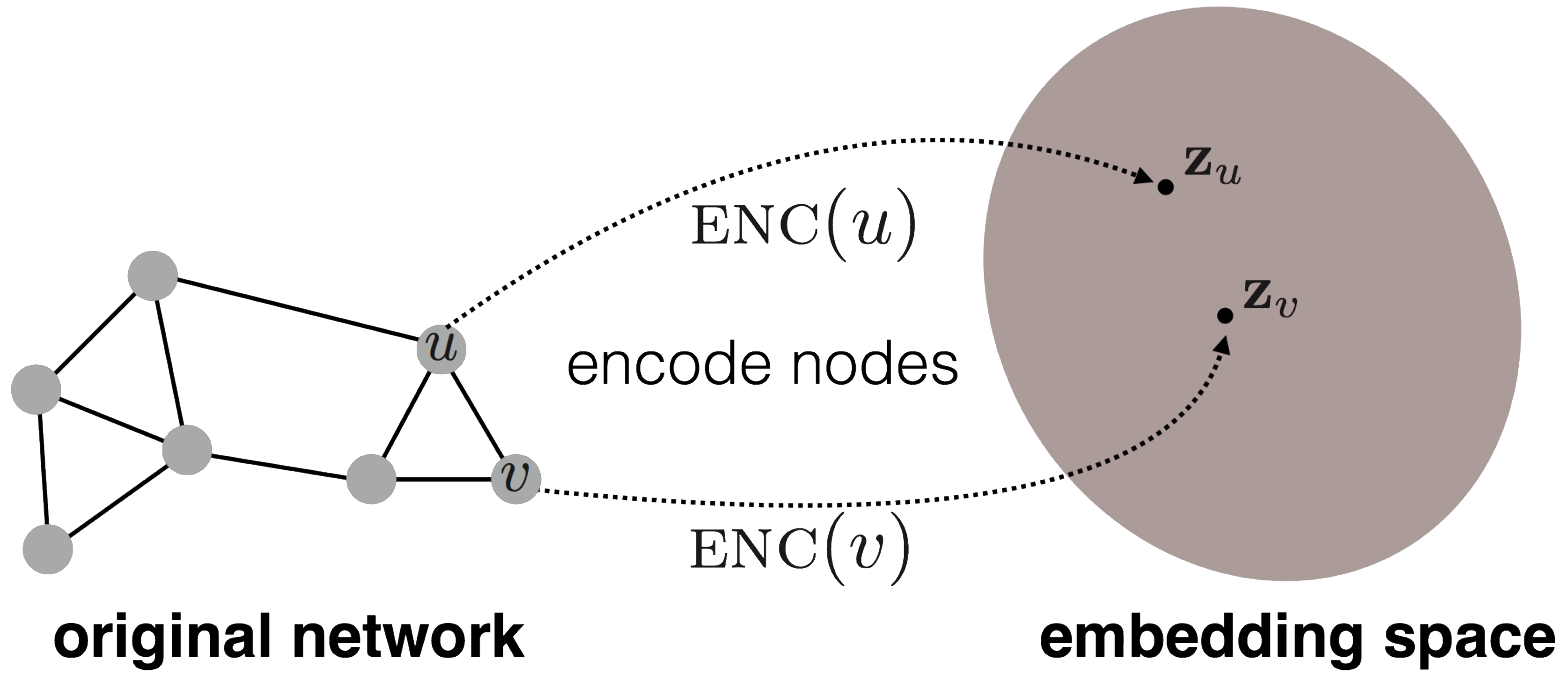


V: {1, 2, 3, 4}

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

Node Embedding

- Goal is to encode nodes so that similarity in the embedding space (e.g., dot product) approximates similarity in the graph



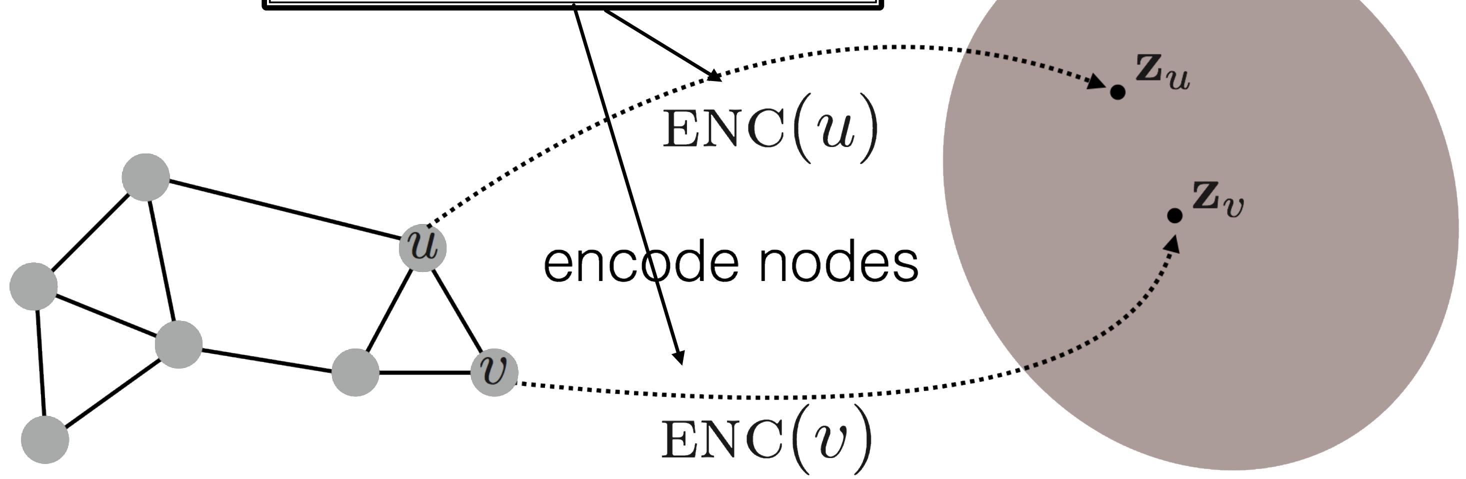
Node Embedding

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

in the original network

Similarity of the embedding

Need to define!



original network **embedding space**

Node Embedding: Key Components

- **Encoder:** maps each node to a low-dimensional vector

$$\mathrm{ENC}(v) = \boxed{\mathbf{z}_v}$$

d-dimensional embedding

node in the input graph

- **Similarity function:** specifies how the relationships in vector space map to the relationships in the original network

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

Decoder

Similarity of u and v in the original network

dot product between node embeddings

“Shallow” Encoding

Simplest encoding approach: **Encoder is just an embedding-lookup**

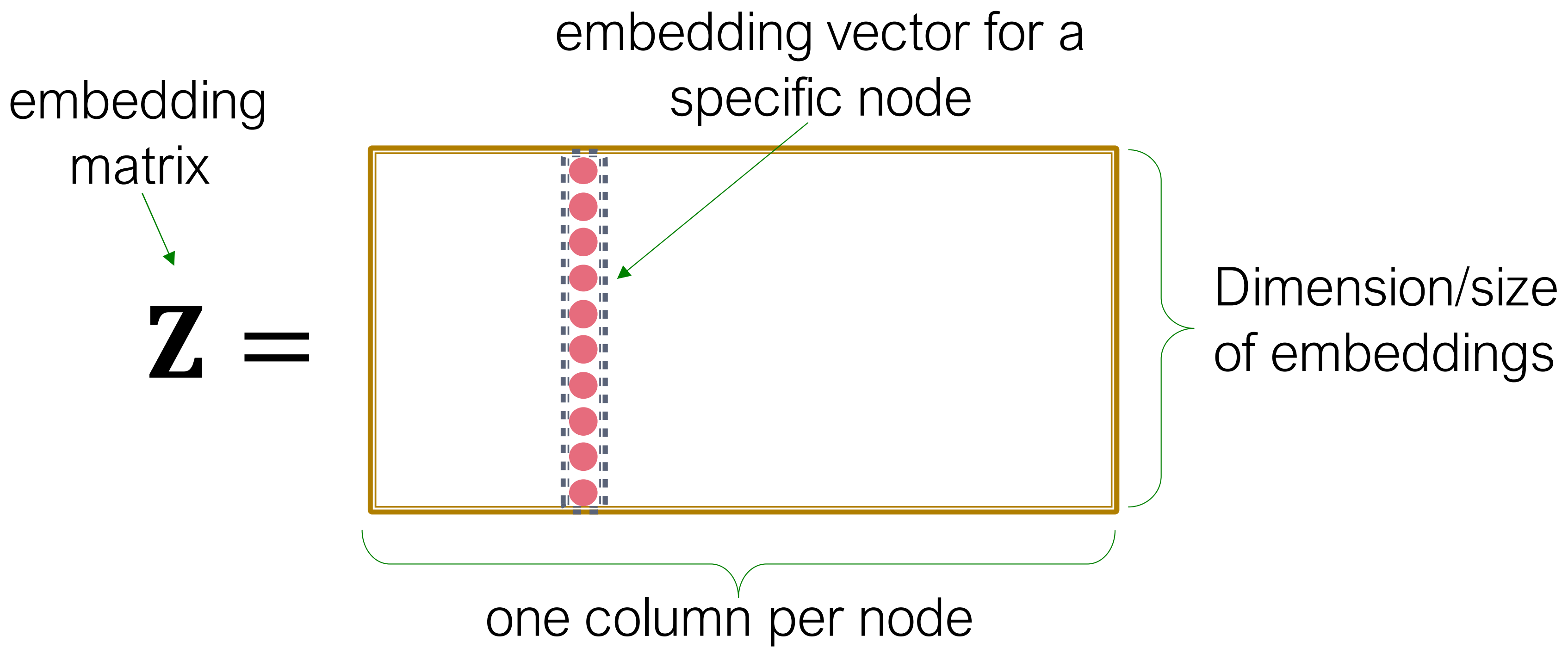
$$\mathrm{ENC}(v) = \mathbf{z}_v = \mathbf{Z} \cdot v$$

$\mathbf{Z} \in \mathbb{R}^{d \times |\mathcal{V}|}$ matrix, each column is a node embedding [what we learn / optimize]

$v \in \mathbb{I}^{|\mathcal{V}|}$ indicator vector, all zeroes except a one in column indicating node *v*

"Shallow" Encoding

Simplest encoding approach: **encoder is just an embedding-lookup**



embedding matrix
$\mathbf{Z} =$
embedding vector for a specific node
Dimension/size of embeddings
one column per node

“Shallow” Encoding

Simplest encoding approach: **Encoder is just an embedding-lookup**

Each node is assigned a unique embedding vector
(i.e., we directly optimize the embedding of each node)

Many methods: DeepWalk, node2vec

Summary

- **Encoder + Decoder Framework**
  - Shallow encoder: embedding lookup
  - Parameters to optimize: $\mathbf{Z}$ which contains node embeddings $\mathbf{z}_u$ for all nodes $u \in V$
  - We will cover deep encoders in the GNNs

  - **Decoder:** based on node similarity.
  - **Objective:** maximize $\mathbf{z}_v^{\mathrm{T}}\mathbf{z}_u$ for node pairs (u, v) that are **similar**

Discussion: How to Define Node Similarity?

- Key choice of methods is **how they define node similarity.**
- Should two nodes have a similar embedding if they…
  - are linked?
  - share neighbors?
  - have similar “structural roles”?

Questions?