Machine learning on graphs - Introduction

Dr Dorina Thanou

11.03.2025

Recap from previous class

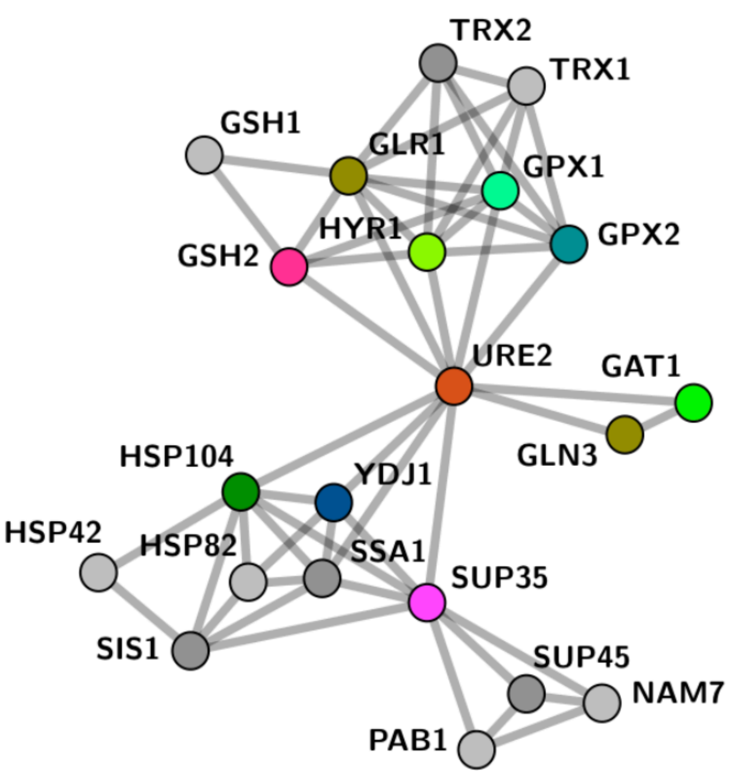
- Networks/graphs are either indicated by the application or constructed from the data

- Spectral graph theory reveals significant properties of the network
  - Spectrum tells us a lot about connectivity, bottlenecks, diameter
  - Eigenvalues provide a notion of frequency
  - Eigenvectors are smooth functions on the graph

- It has applications in network tasks, where preserving geometry is crucial

In the following lectures…

- How can we infer useful information from data that live on a graph?
  - Graphs could be weighted or unweighted
  - Nodes could have attributes



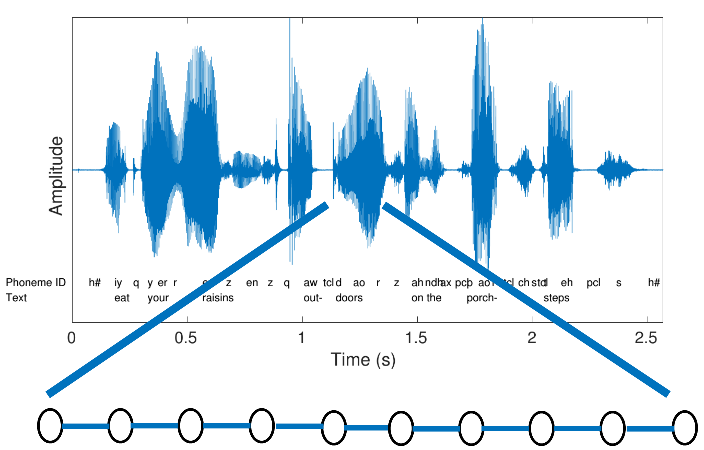
Machine learning (ML)/ Signal processing (SP) on graphs → Actionable knowledge

e.g., node/graph classification, signal inpainting/denoising

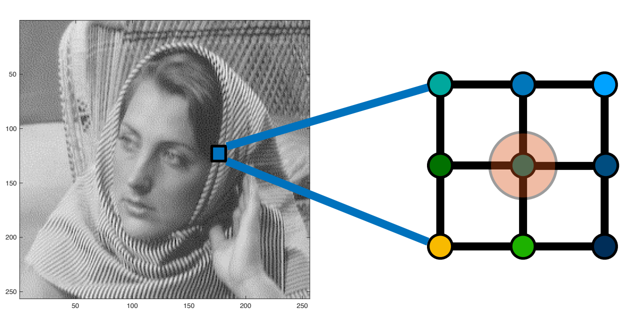
$$X, \mathcal{G} \rightarrow f(X, \mathcal{G}) \rightarrow Y$$

Why learning from graphs is hard?

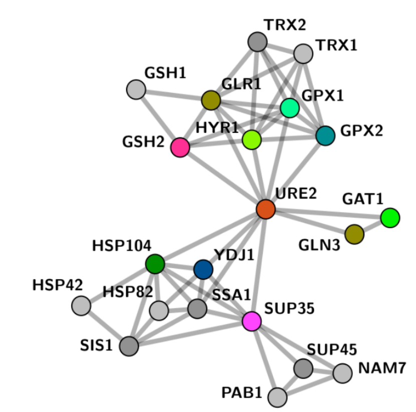
- Contrary to traditional modalities:
  - Graphs capture complex and irregular connections
  - There is no explicit notion of ordering
  - Nodes can have multiple attributes



Domain: line



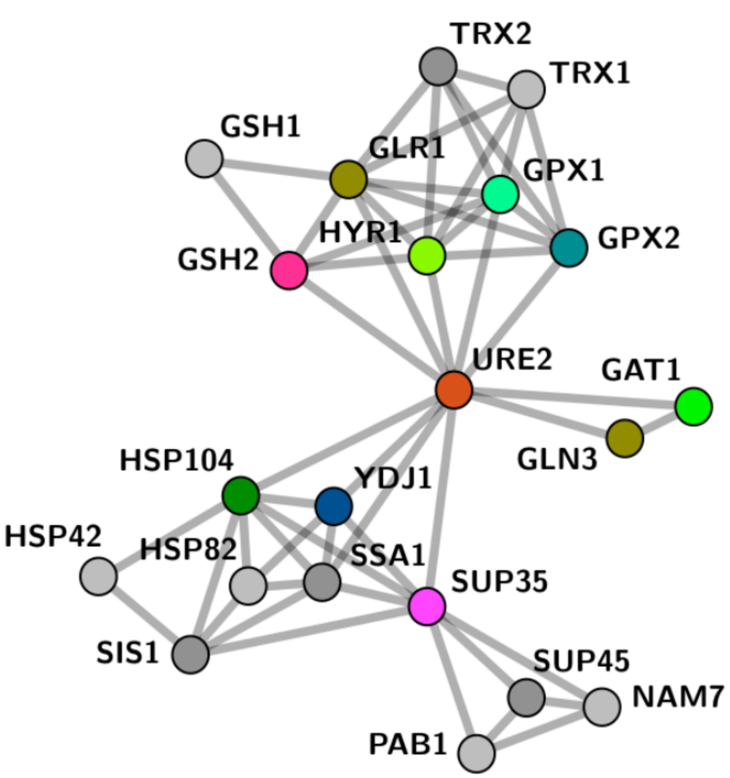
Domain: grid



Domain: irregular graph

Traditional ML pipeline on graphs

- How can we infer useful information from data that live on a graph?
  - Graphs could be weighted or unweighted
  - Nodes could have attributes



→ **Graph-based features** → **ML model** → **Actionable knowledge**

e.g., node/graph classification, signal inpainting/denoising

$$X, \mathcal{G} \rightarrow \phi(X, \mathcal{G}) \rightarrow f(\phi(X, \mathcal{G})) \rightarrow Y$$

Graph-structured features/embeddings: A high level overview

- **Hand-crafted features:** Capture some structural properties of the graph, followed by some statistics (signatures)
- **Graph kernel methods:** Design similarity functions in an embedding space
- **Spectral features:** Capture the graph properties through spectral graph theory

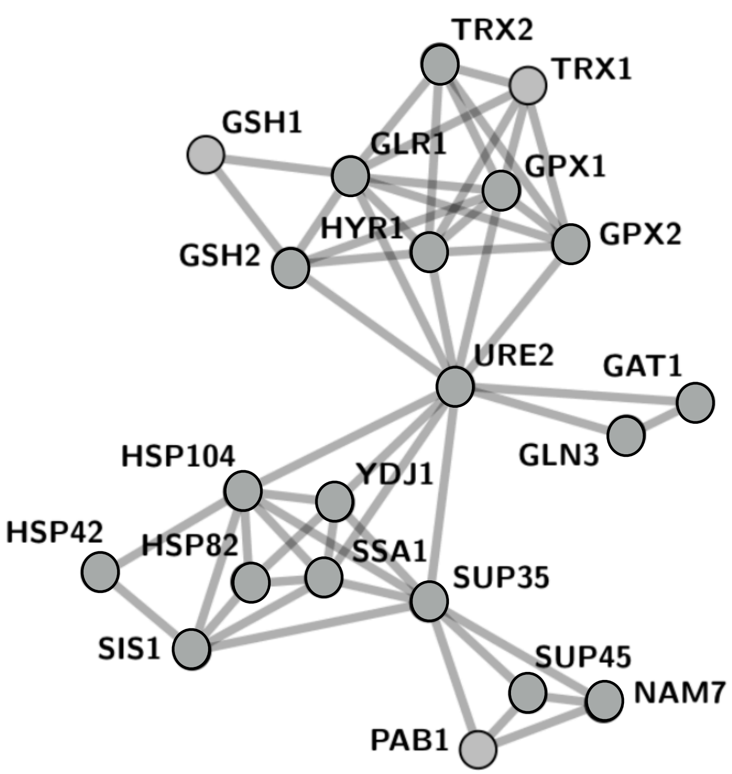
Model-driven

- **Learned features:** Learn graph features directly from data by designing models based on meaningful assumptions
  - **Unsupervised (shallow) embeddings:** Learn features based on different ways of preserving information from the original graph (often without node attributes)
  - **Graph neural network features:** Learn features from the data using a well-designed family of neural networks (often with node attributes)

Data-driven

In this lecture

- How can we infer useful information **only from the graph structure**?



Graph-based features → **ML model** → **Actionable knowledge**

e.g., node/graph classification, signal inpainting/denoising

Feature engineering

$$\mathcal{G} \rightarrow \phi(\mathcal{G}) \rightarrow f(\phi(\mathcal{G})) \rightarrow Y$$

Graph-structured features/embeddings: A high level overview

- **Hand-crafted features:** Capture some structural properties of the graph, followed by some statistics (signatures)
- **Graph kernel methods:** Design similarity functions in an embedding space
- **Spectral features:** Capture the graph properties through spectral graph theory

Model-driven

- **Learned features:** Learn graph features directly from data by designing models based on meaningful assumptions
  - **Unsupervised (shallow) embeddings:** Learn features based on different ways of preserving information from the original graph (often without node attributes)
  - **Graph neural network features:** Learn features from the data using a well-designed family of neural networks (often with node attributes)

Data-driven

Following lectures

Outline

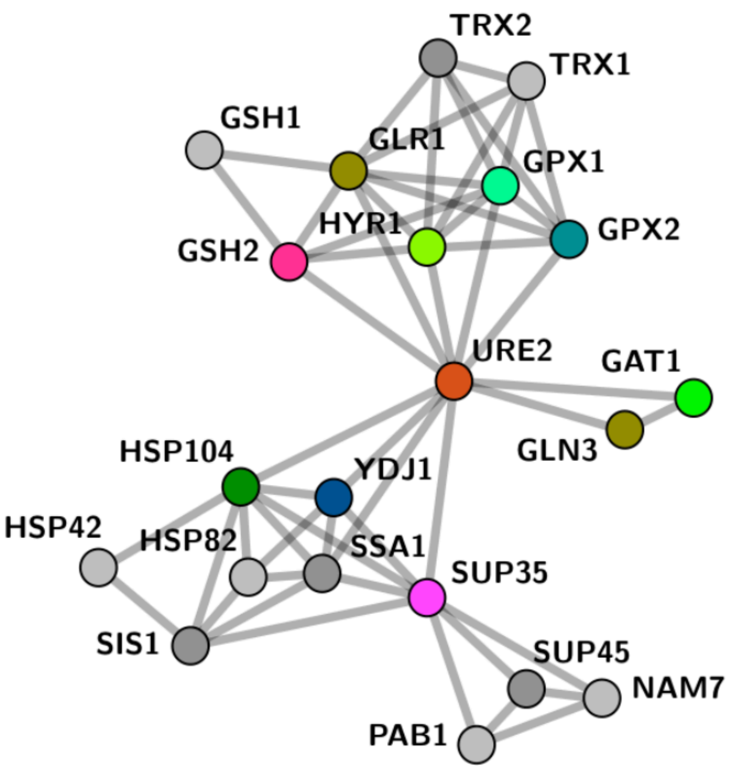
- Machine learning pipeline on graphs
- Traditional graph structural features
  - Node level tasks
  - Graph level tasks
  - Edge level tasks

Outline

- **Machine learning pipeline on graphs**

- Traditional graph structural features
  - Node level tasks
  - Graph level tasks
  - Edge level tasks

Traditional ML pipeline: Input



$X, \mathcal{G}$

- Input:
  - Graph: $\mathcal{G} = (\mathcal{V}, \mathcal{E}, W)$
  - Graph with attributes: $\mathcal{G}, X$

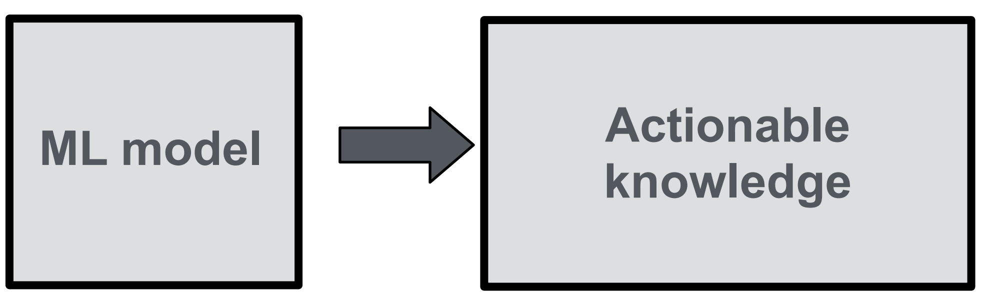
Traditional ML pipeline: Features

Graph-based features

$$\phi(X, \mathcal{G})$$

- Should reveal important information regarding the graph structure
- Key to achieving good model performance
- Features can be defined at different scales
  - At a node, edge, sets of nodes, entire graph level
- The choice of the features depends on
  - the end task
  - prior knowledge on the data

Traditional ML pipeline: Learning tasks

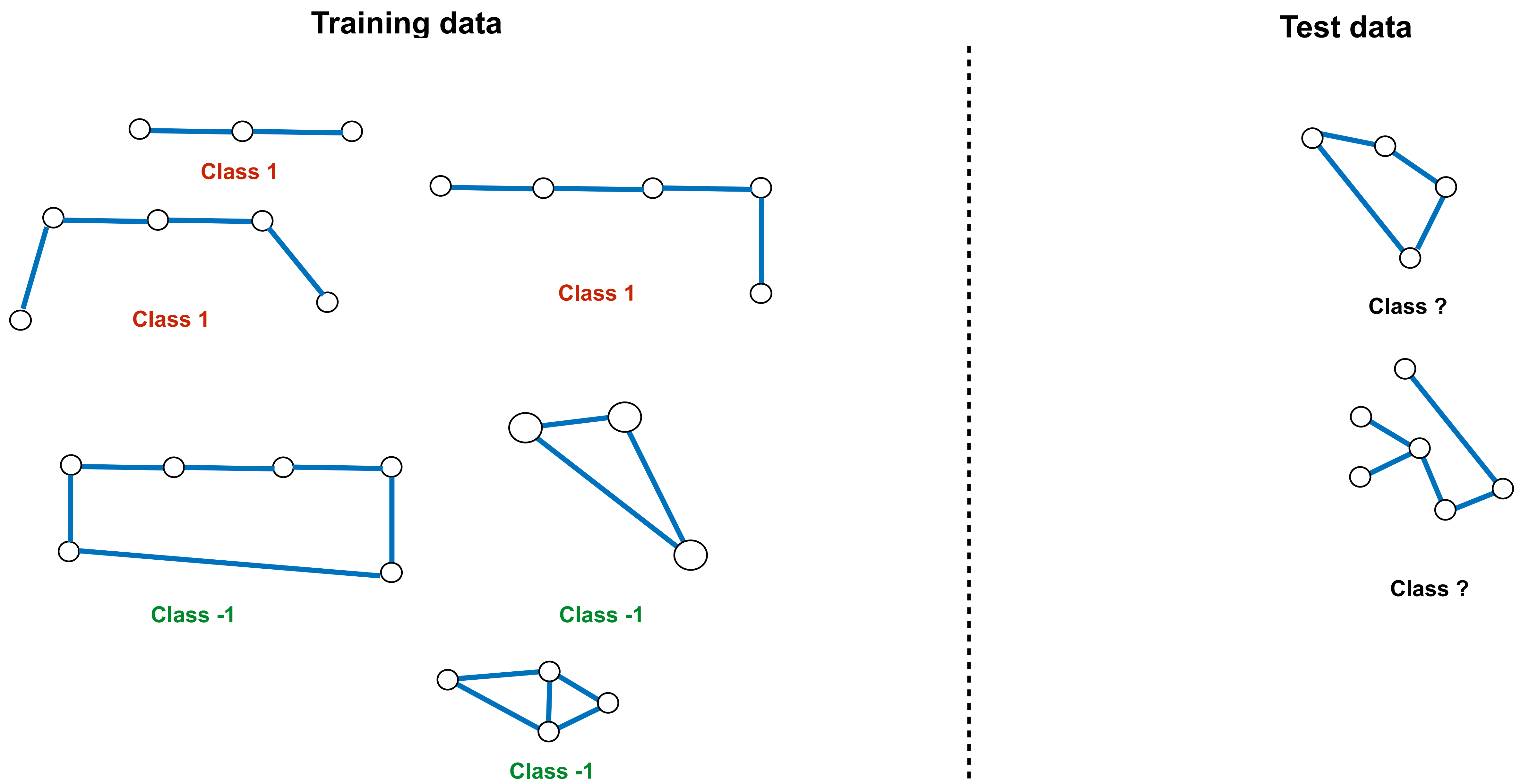


e.g., node/graph classification, signal inpainting/denoising

$$f(\phi(X,\mathcal{G})) \rightarrow Y$$

- The features are given as input to an ML model
  - Examples: logistic regression, SVM, neural networks, etc.
- Training phase:
  - Given a set of graph-based features, train a model f that predicts the correct Y
- Testing phase:
  - Given a new node/link/graph, compute its features, and give them as an input to f to make a prediction

Illustrative example: Graph classification



What are the key features for classifying my graphs?

Illustrative example: Graph classification with SVM

- Classical SVM setup:
  - Given a set of M training graphs, across with their class labels $\mathcal{D} = \{(\mathcal{G}_i, y_i)\}_{i=1}^{M}$ learn a classifier that predicts the label of a new graph

$$\max_{\alpha} \sum_{i=1}^{M} \alpha_i - \frac{1}{4} \sum_{i,j=1}^{M} \alpha_i \alpha_j y_i y_j < \phi(\mathcal{G}_i), \phi(\mathcal{G}_j) >$$

Graph features or embeddings

$$\text{subject to } \sum_{i=1}^{N} \alpha_i y_i = 0$$

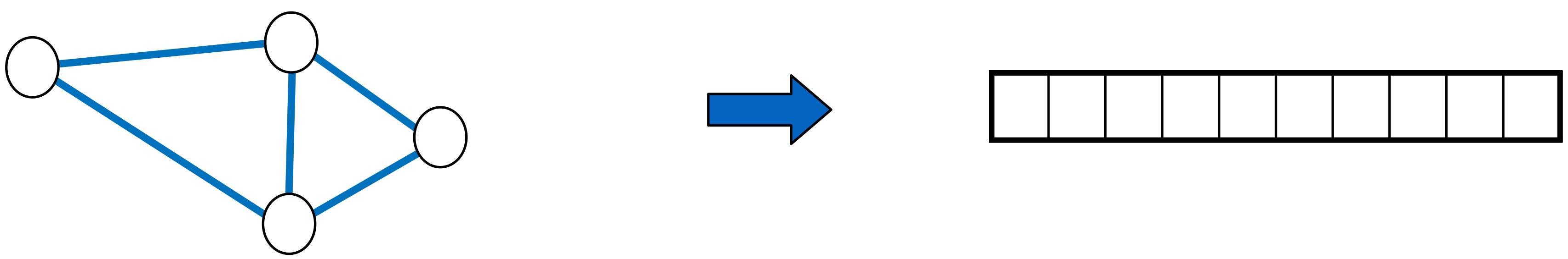
  - Use the learned model to classify new graph instances

How do we compute graph features?

[Scholkopf et al., Learning with kernels, MIT Press, 2002]

Highly challenging for ML

- Features (often know as embeddings) should capture the intrinsic structure of the graph
- Most ML algorithms require features to be represented as a fixed length feature vector



?

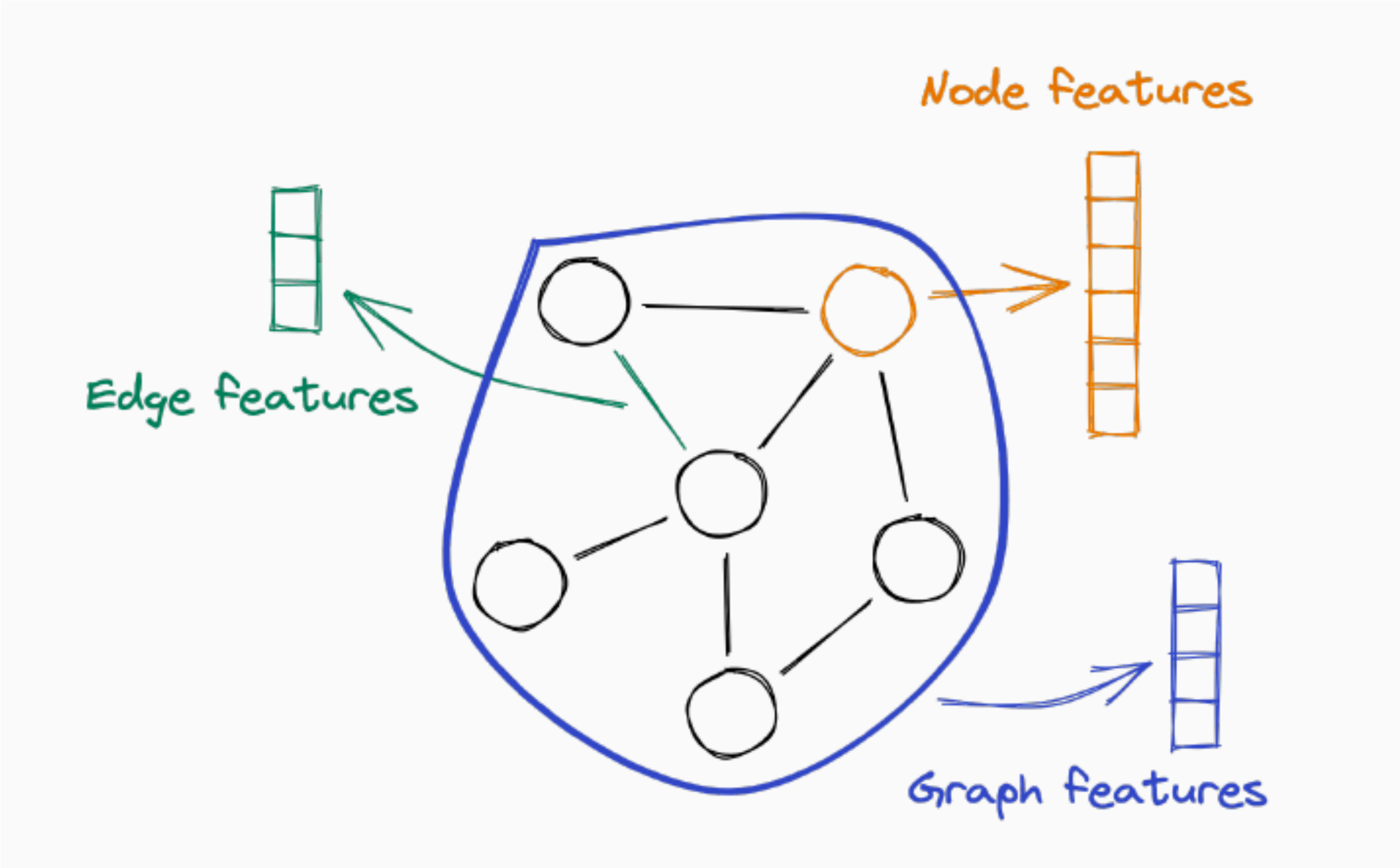
- **Todays' focus:**
  - Graphs without node attributes, i.e., inferring information **only from the graph structure** (graph structural features)
  - **Hand-designed features**, i.e., features that are designed based on some priors (no learning involved!)

Outline

- Machine learning pipeline on graphs

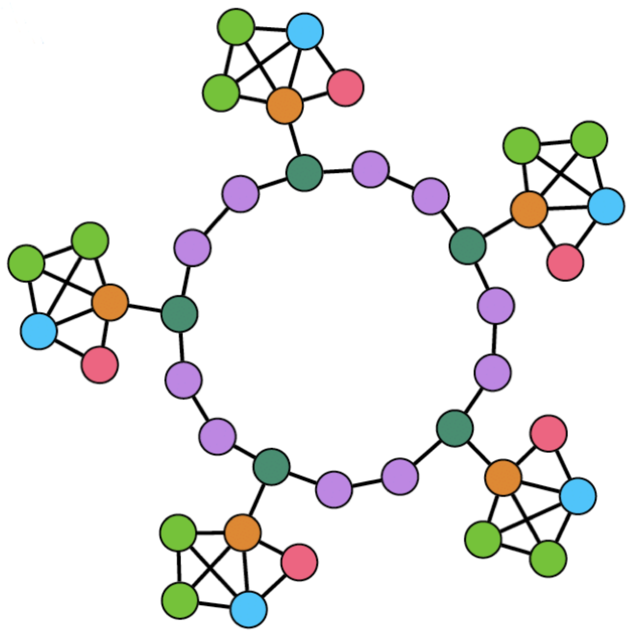
- **Traditional graph structural features**
  - Node level tasks
  - Graph level tasks
  - Edge level tasks

Extracting structural information at different levels



Node level features

- Typically useful for node classification/clustering tasks



Color reflects the degree!

- Aim at characterizing the structure and position of a node in the network

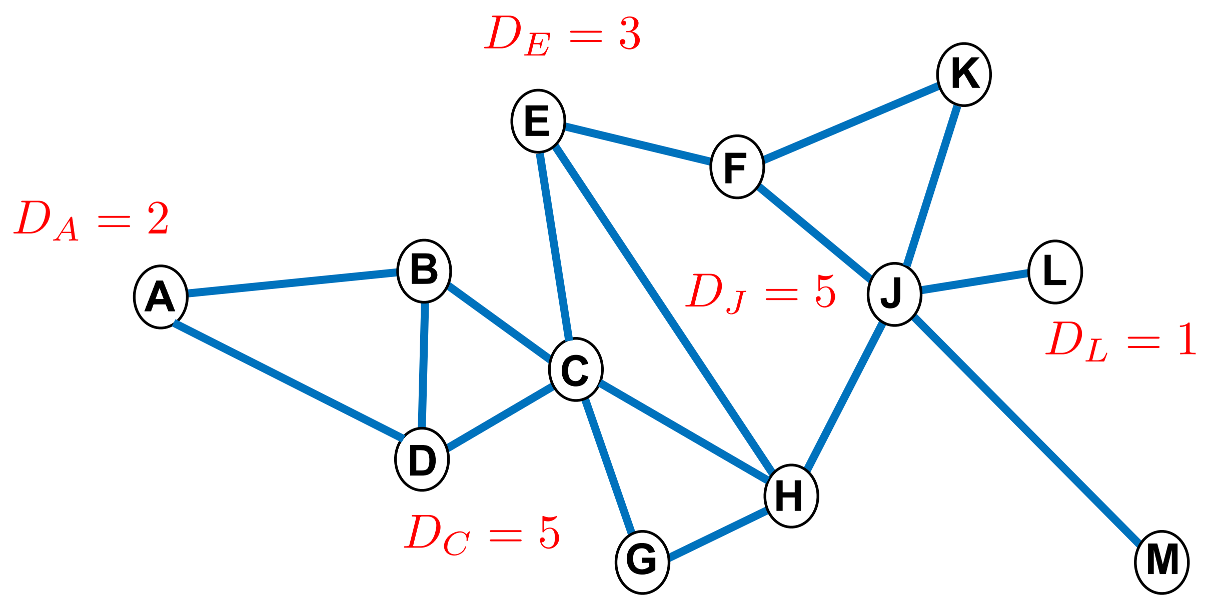
Common node level features

- Node degree
- Node centrality
- Clustering coefficient
- Graphlets

Node degree

- The degree D_u of node u is the number of edges (neighboring nodes) the node has

$$D_u = \sum_{v \in \mathcal{N}_u} W_{vu}$$



- Usually normalized with the maximum number of nodes $\tilde{D}_u = \dfrac{D_u}{|\mathcal{V}|}$
- The node degree feature treats all nodes equally

Node centrality

- Node centrality takes the node importance in a graph into account

- Various ways to model importance

  - Betweenness centrality

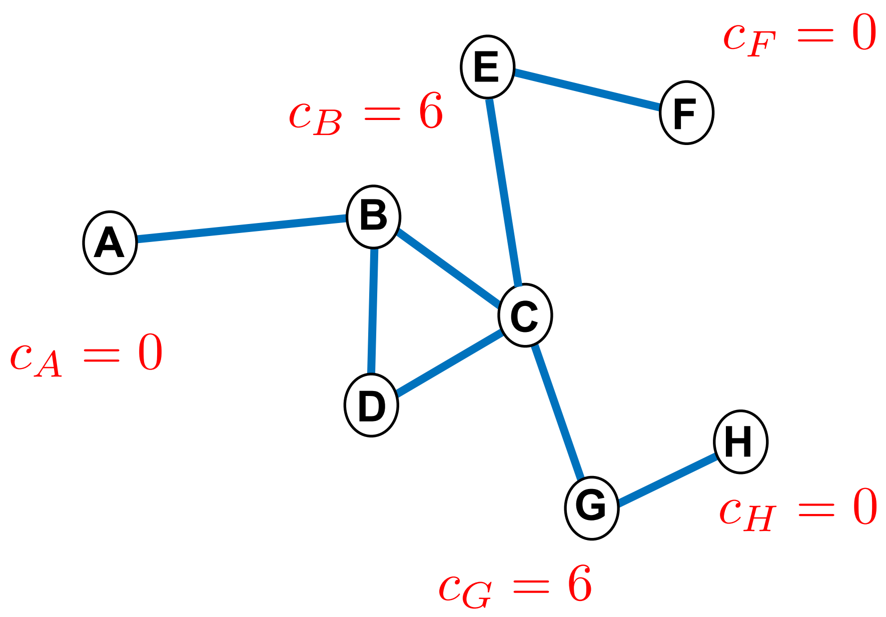
  - Closeness centrality

  - Eigenvector centrality

Betweenness centrality

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

$$c_u = \sum_{v \neq u \neq z} \frac{\#(\text{shortest paths between } v \text{ and } z \text{ that contains } u)}{\#(\text{shortest paths between } v \text{ and } z)}$$



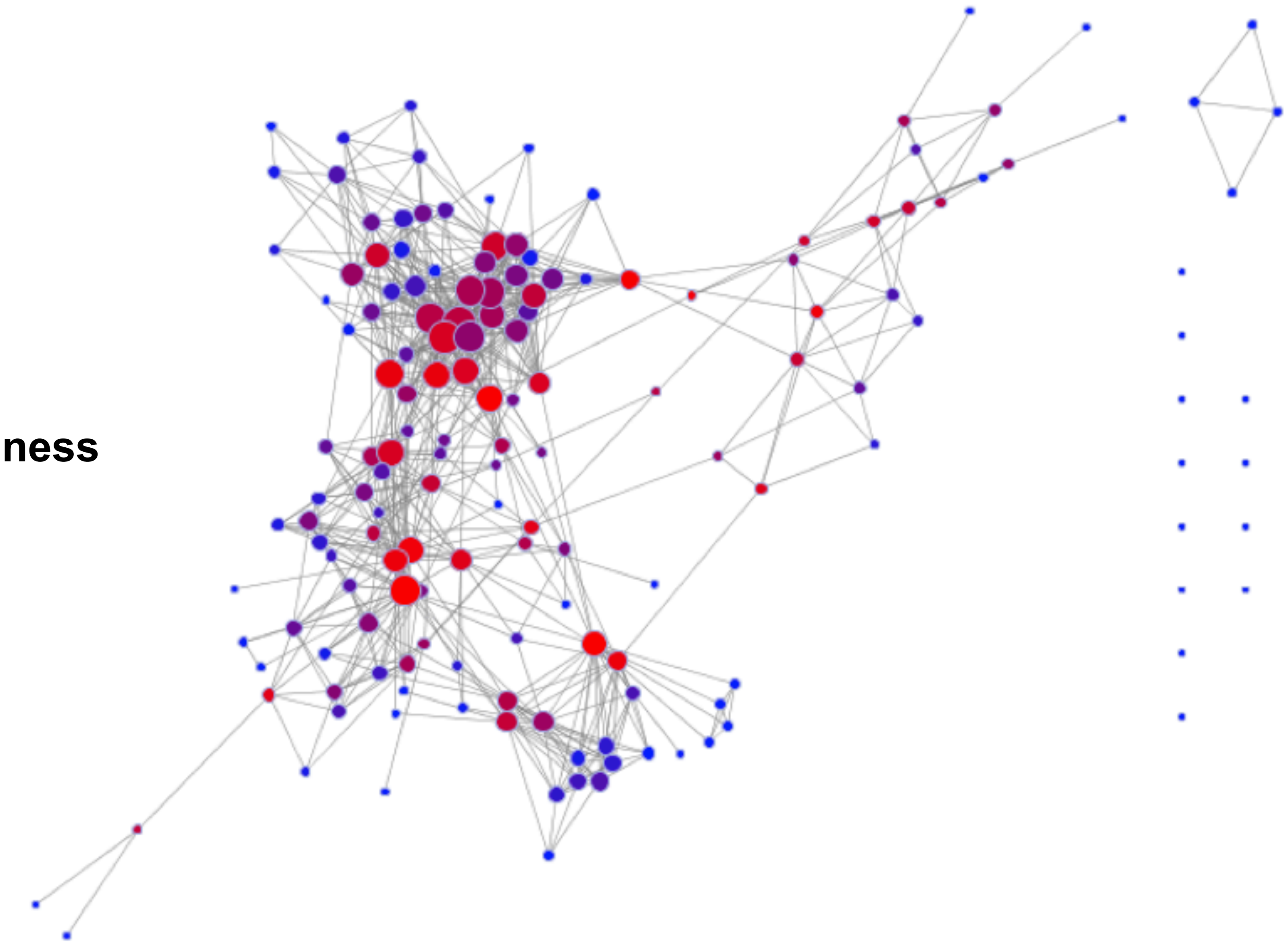
Example:

$c_B = 6$

$(ABD, ABC, ABCE, ABCEF, ABCG, ABCGH)$

Comparison between degree and betweenness centrality

Color: Betweenness
Size: Degree

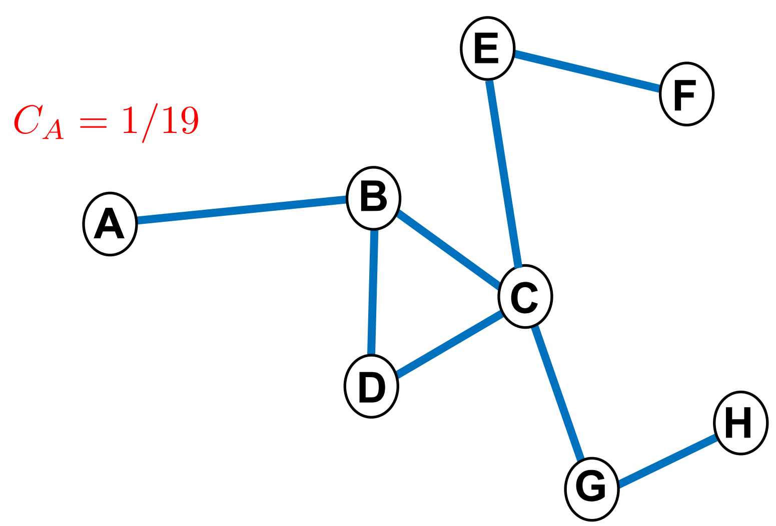


Degree is high if a node has many direct connections (e.g., friends)
Betweenness is high if a node is 'between' other nodes

Closeness centrality

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

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

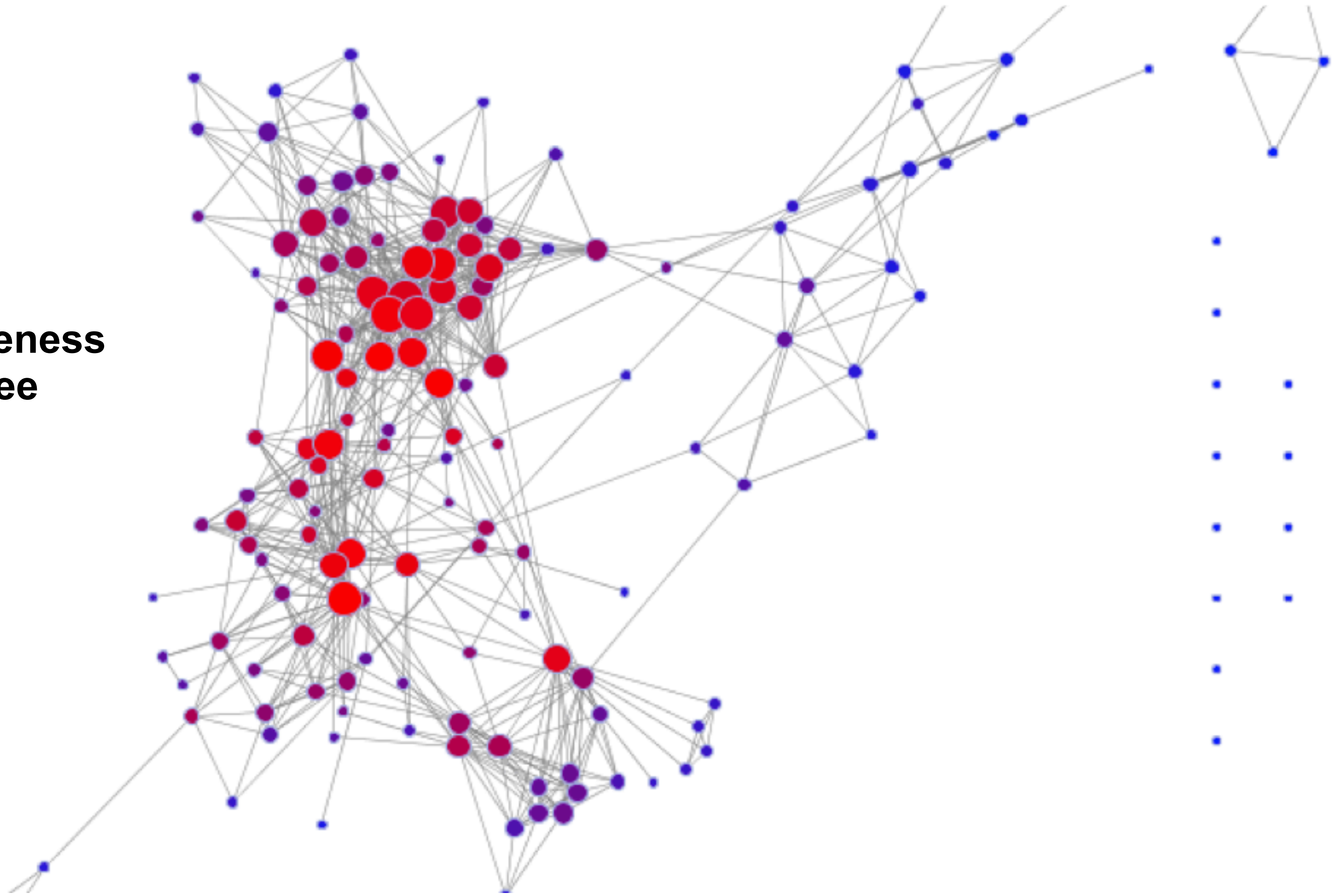


Example:

$$c_A = 1/(1+2+2+3+4+3+4) = 1/19$$
$$(AB, ABC, ABD, ABCE, ABCEF, ABCG, ABCGH)$$

Comparison between degree and closeness centrality

Color: Closeness
Size: Degree



Degree is high if a node has many direct connections (e.g., friends)
Closeness is high if a node is in the 'middle' of things

Eigenvector centrality

- A node u is important if it is surrounded by important neighbors

$$c_u = \frac{1}{\lambda} \sum_{v \in \mathcal{N}_u} W_{uv} c_v$$

- It can be written as the eigenvector equation

$$\lambda c = W c$$

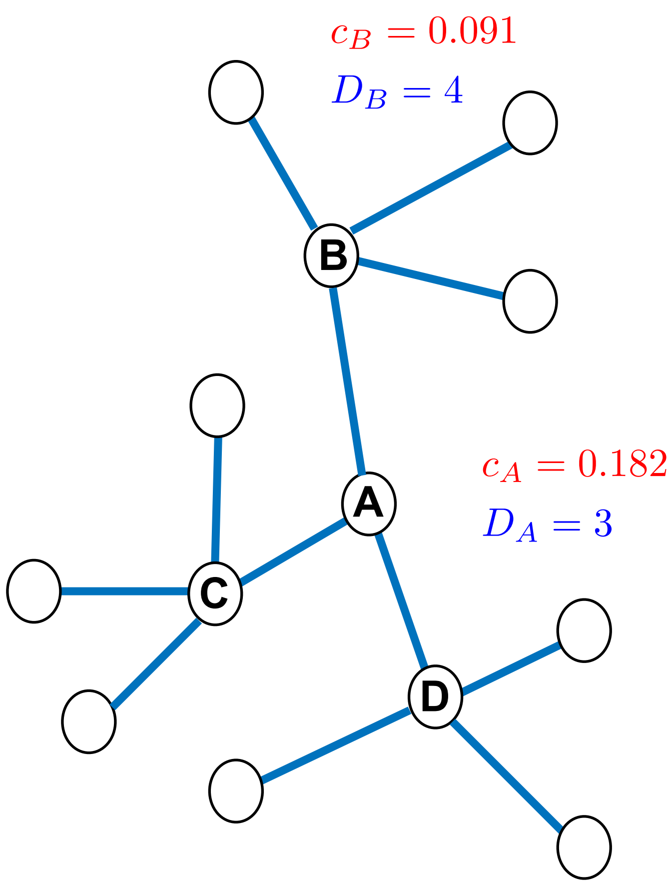
- Centrality measure: (dominant) eigenvector corresponding to the largest eigenvalue

- Can be computed using power iteration

$$c^{t+1} = W c^t$$

Contains the number of length t+1 paths arriving at each node!

Example of eigenvector centrality



Degree is high if a node has many direct neighbors
Centrality is high if a node has well-connected neighbors

(Global) Clustering coefficient

- The clustering coefficient characterizes the subgraph containing the neighbors of a node, and all edges between nodes in its neighborhood

- It measures how tightly clustered a node's neighborhood is

$$cc_u = \frac{|(v_1, v_2) \in \mathcal{E} : v_1, v_2 \in \mathcal{N}_u|}{\binom{D_u}{2}}$$

- It is computed as the proportion of closed triangles in a node's local neighborhood
  - It represents the probability that two neighbors of a node are linked to each other (see previous lecture on graph theory basics)

(Global) Clustering coefficient

- The clustering coefficient characterizes the subgraph containing the neighbors of a node, and all edges between nodes in its neighborhood

- It measures how tightly clustered a node's neighborhood is

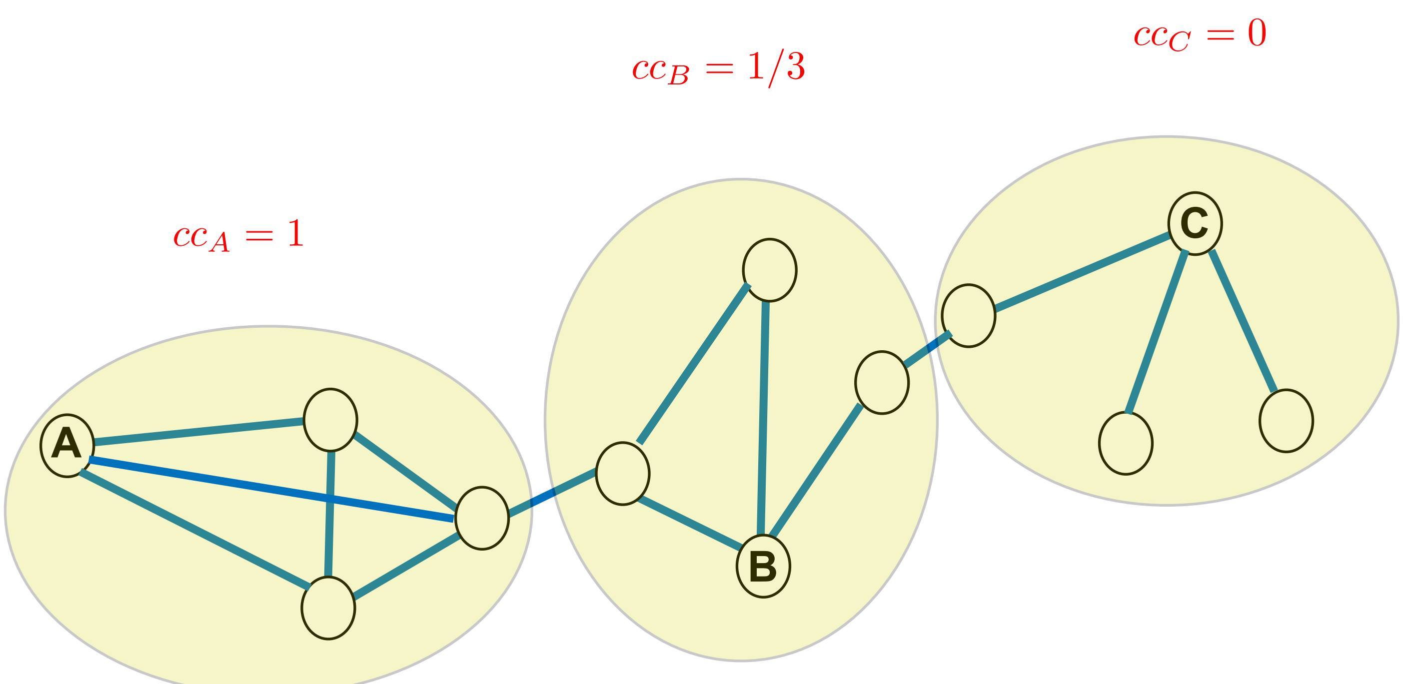
edges among neighboring nodes

$$cc_u = \frac{\boxed{|(v_1, v_2) \in \mathcal{E} : v_1, v_2 \in \mathcal{N}_u|}}{\boxed{\binom{D_u}{2}}}$$

node pairs among neighboring nodes

- It is computed as the proportion of closed triangles in a node's local neighborhood
  - It represents the probability that two neighbors of a node are linked to each other (see previous lecture on graph theory basics)

Example of clustering coefficient

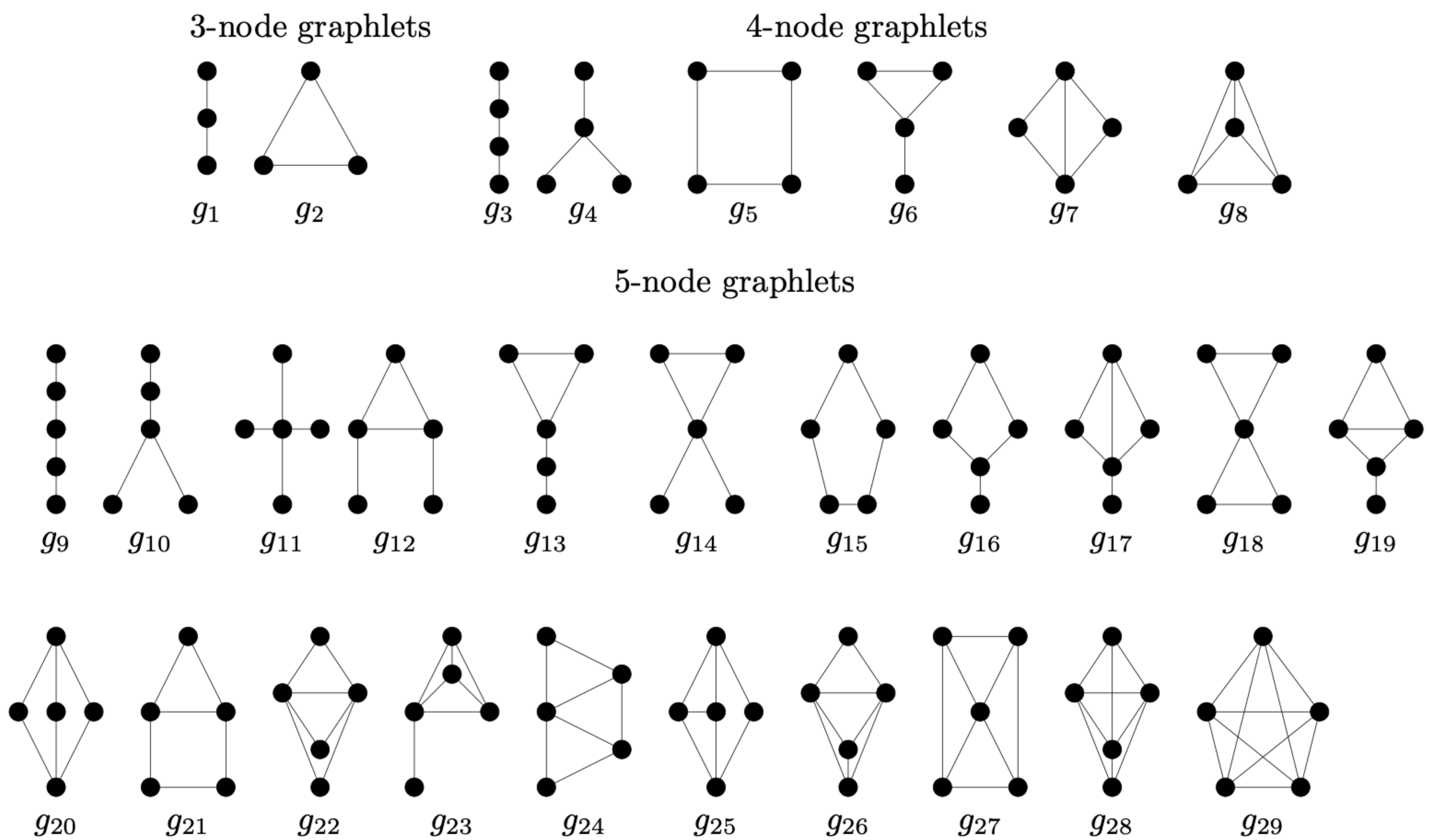


Well-connected neighborhood

Sparsely connected neighborhood

Graphlets

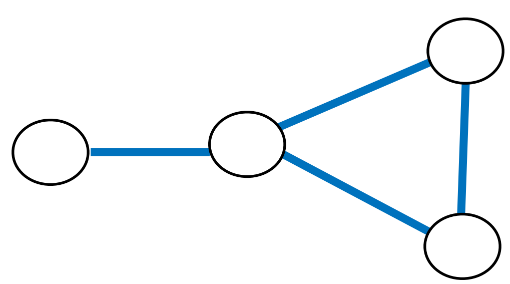
- Small subgraphs that describe the structure of node network neighborhood



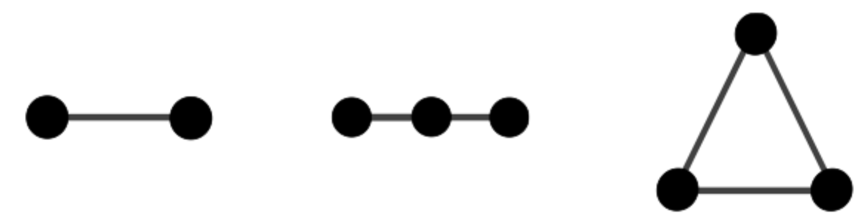
- These topological structures can be used to define a frequency histogram

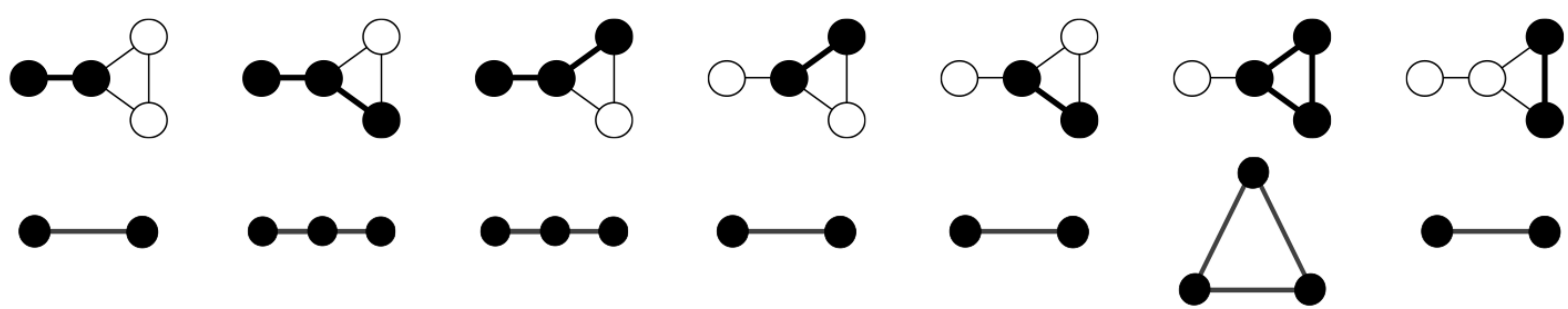
Example of graphlets

- Graphlet Degree Vectors (GDV): counts the number of graphlets that a node belongs to



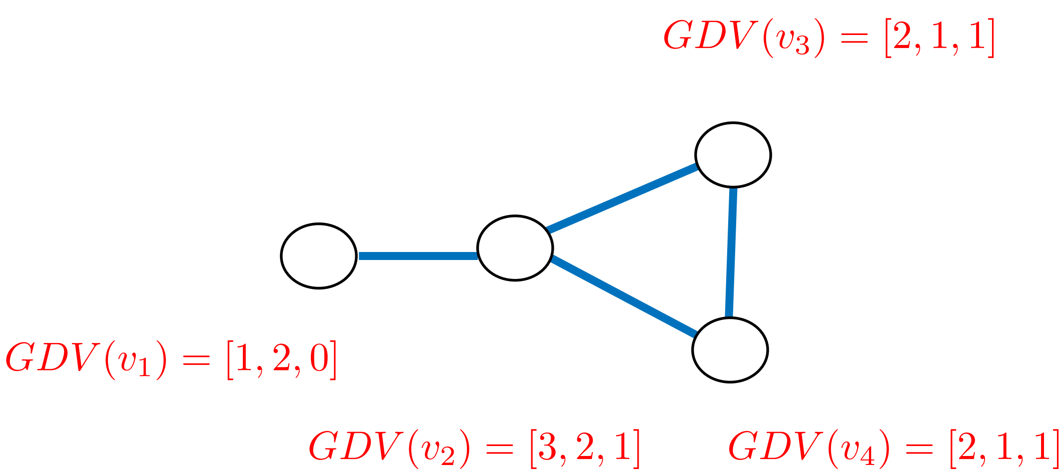
Possible graphlets:



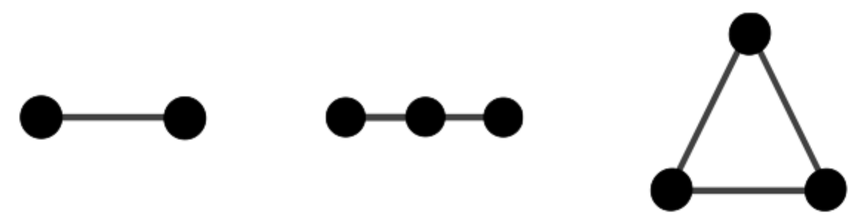


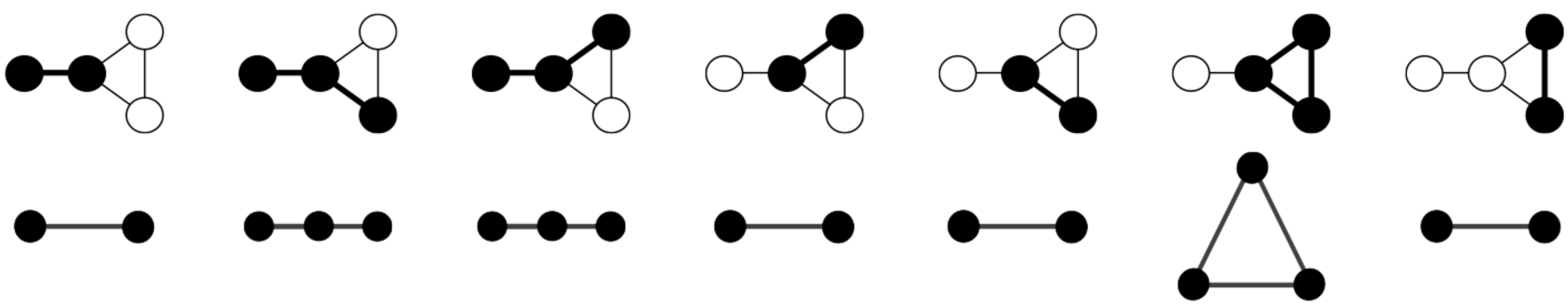
Example of graphlets

- Graphlet Degree Vectors (GDV): counts the number of graphlets that a node belongs to



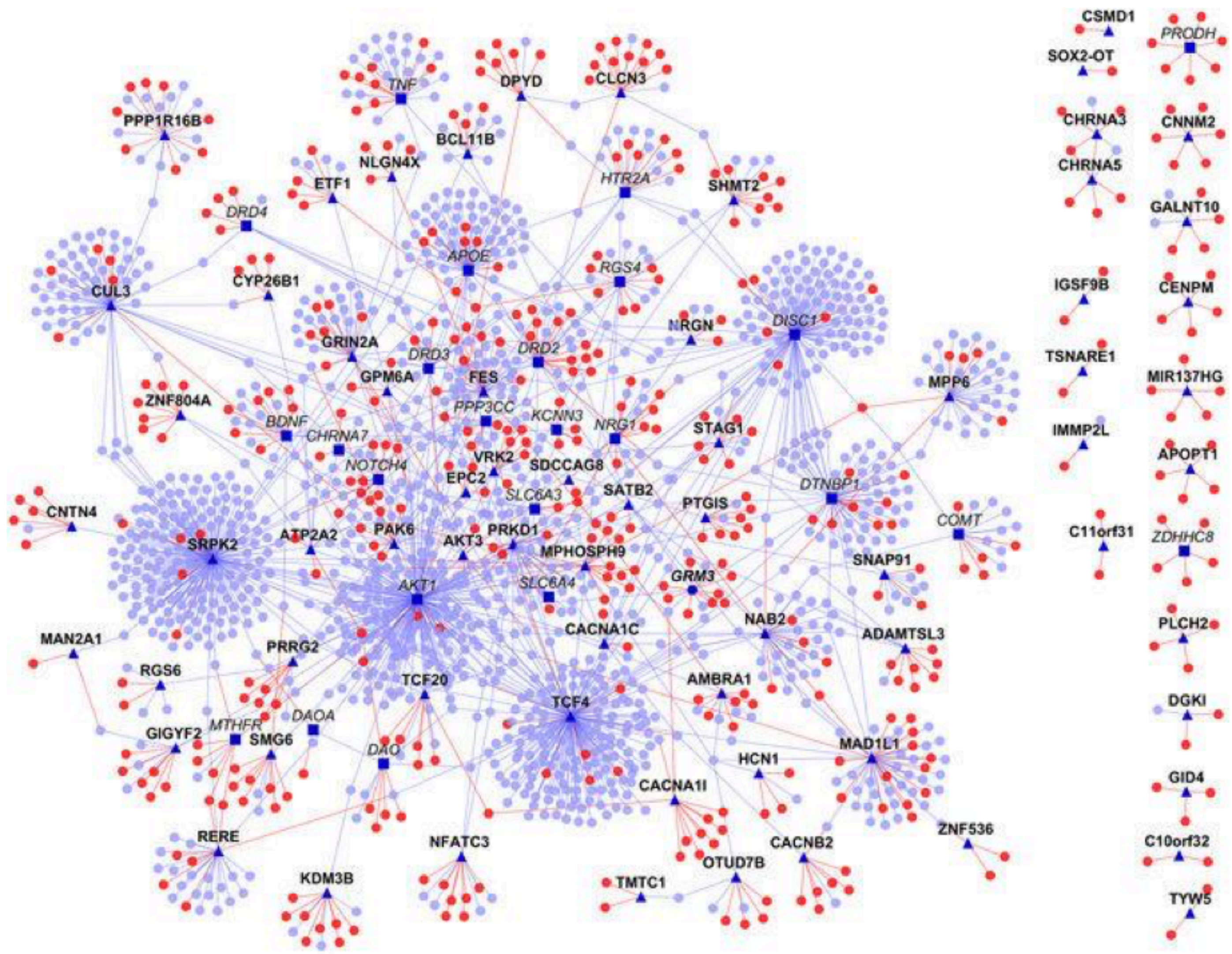
Possible graphlets:





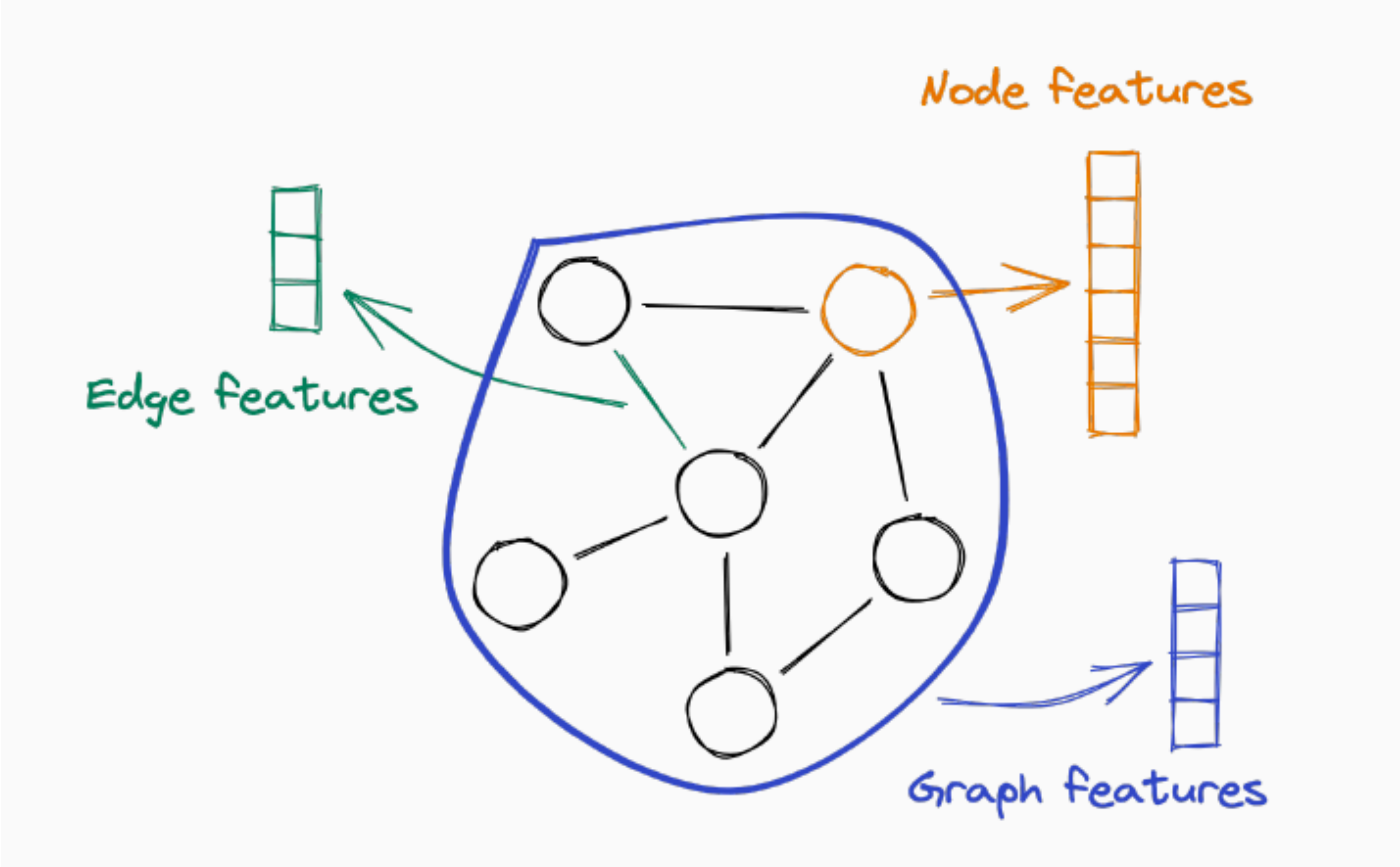
Graphlets for protein-protein interactions

- Often used in classifying function of proteins in the interactome



[Ganapathiraju et al. 2016. Schizophrenia interactome with 504 novel protein–protein interactions. Nature]

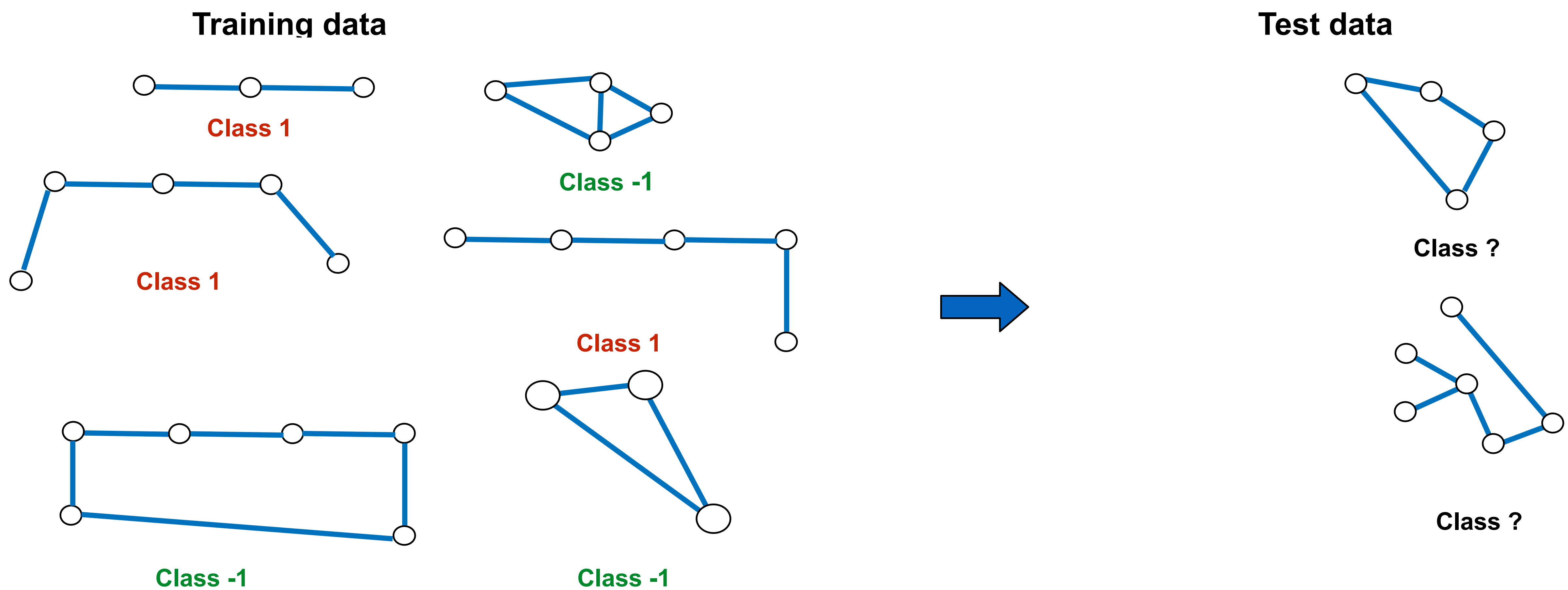
From node level to graph level task



How can we design features that characterize the structure of the entire graph?

Illustrative example: Graph classification

- Common assumption: Graphs with similar structure have similar label

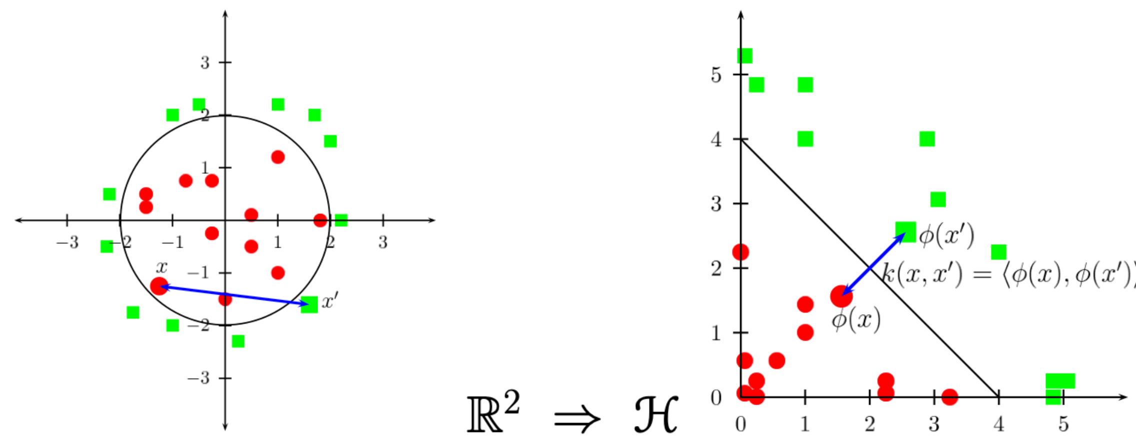


What is a good similarity metric between graphs?

Kernels in a nutshell

- **Intuition:** Move the learning task to a feature space where the task is easier

- Usually a two steps approach:
  - Map objects x and x' via mapping ϕ to $\mathcal{H}$
  - Measure the similarity in the feature space $\langle \phi(x), \phi(x') \rangle$



- Kernel trick: compute the inner product in $\mathcal{H}$ as kernel in the input space

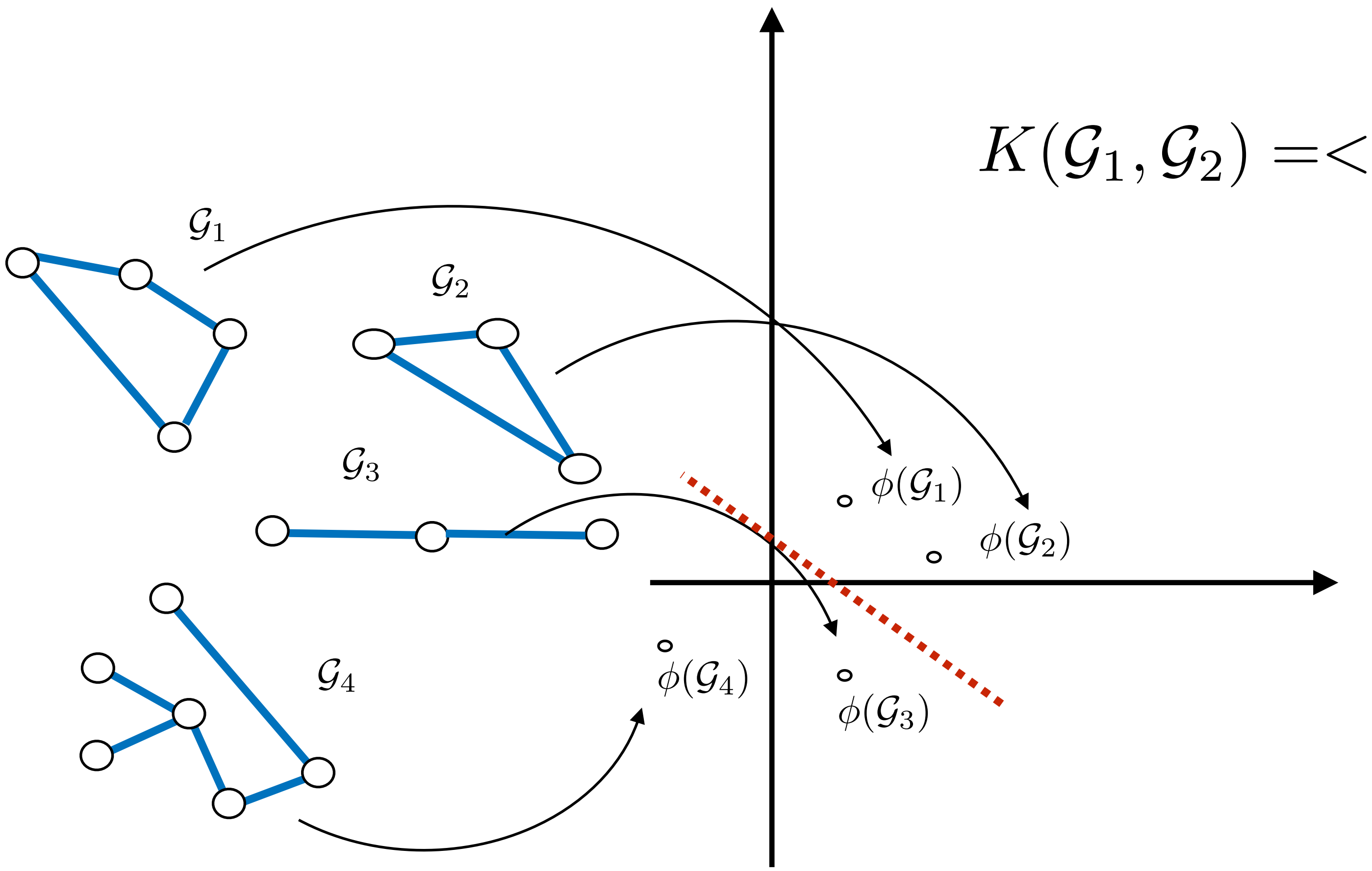
$$K(x, x') = \langle \phi(x), \phi(x') \rangle$$

K is a measure of similarity

Graph kernel methods

- Let $\phi(\mathcal{G}_1), \phi(\mathcal{G}_2)$ be feature representations of graphs $\mathcal{G}_1, \mathcal{G}_2$ in a very high dimensional feature space
- Define functions/kernels which measure the similarity between graphs

$$K(\mathcal{G}_1, \mathcal{G}_2) =< \phi(\mathcal{G}_1), \phi(\mathcal{G}_2) >$$



- Provide kernels as an input to a classifier: e.g., SVM

Illustrative example: Graph classification with Kernel SVM

- Classical SVM setup:
  - Given a set of M training graphs, across with their class labels $\mathcal{D} = \{(\mathcal{G}_i, y_i)\}_{i=1}^{M}$ learn a classifier that predicts the labels of a new graph

$$\max_{\alpha} \sum_{i=1}^{M} \alpha_i - \frac{1}{4} \sum_{i,j=1}^{M} \alpha_i \alpha_j y_i y_j K(\mathcal{G}_i, \mathcal{G}_j)$$

Graph kernel

$$\text{subject to } \sum_{i=1}^{N} \alpha_i y_i = 0$$

  - Use the learned model to classify new graph instances

How do we compute graph features/kernels?

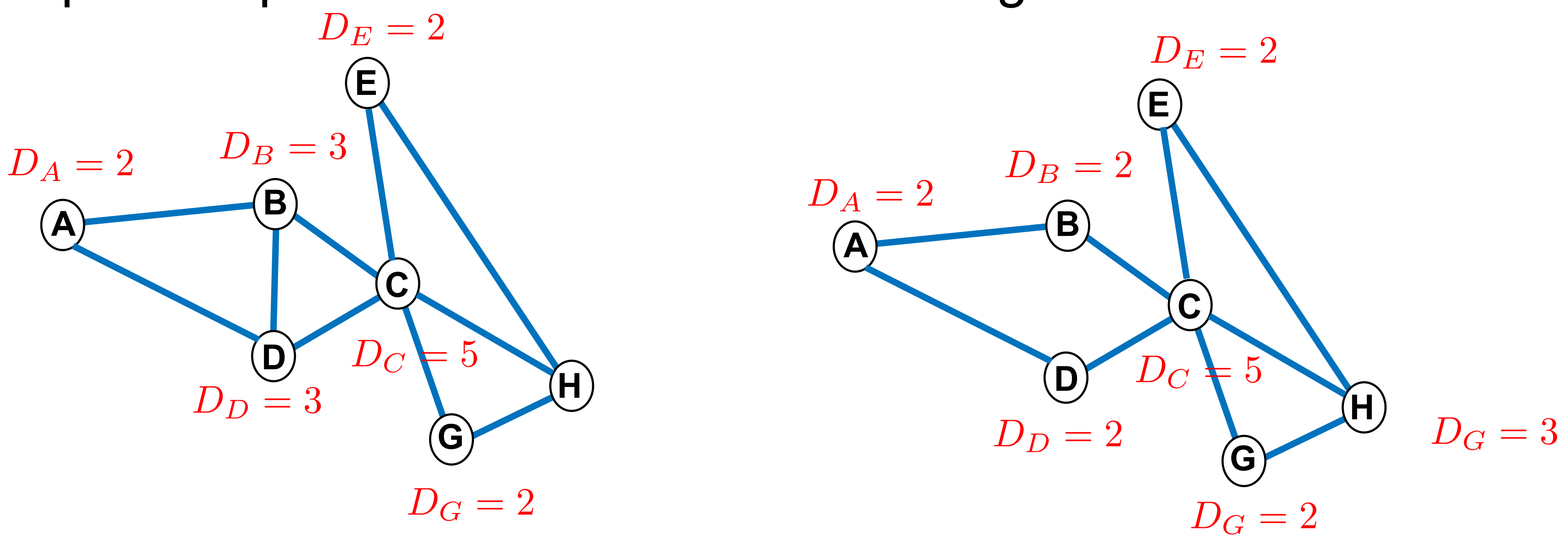
Graph level features

- Bag of nodes
- Graphlet kernel
- The Weisfeiler-Lehman kernel

Bag of nodes

- Use node features to compute histograms or other summary statistics to define a graph level representation, i.e., graph features

- Example: Graph features based on node degrees

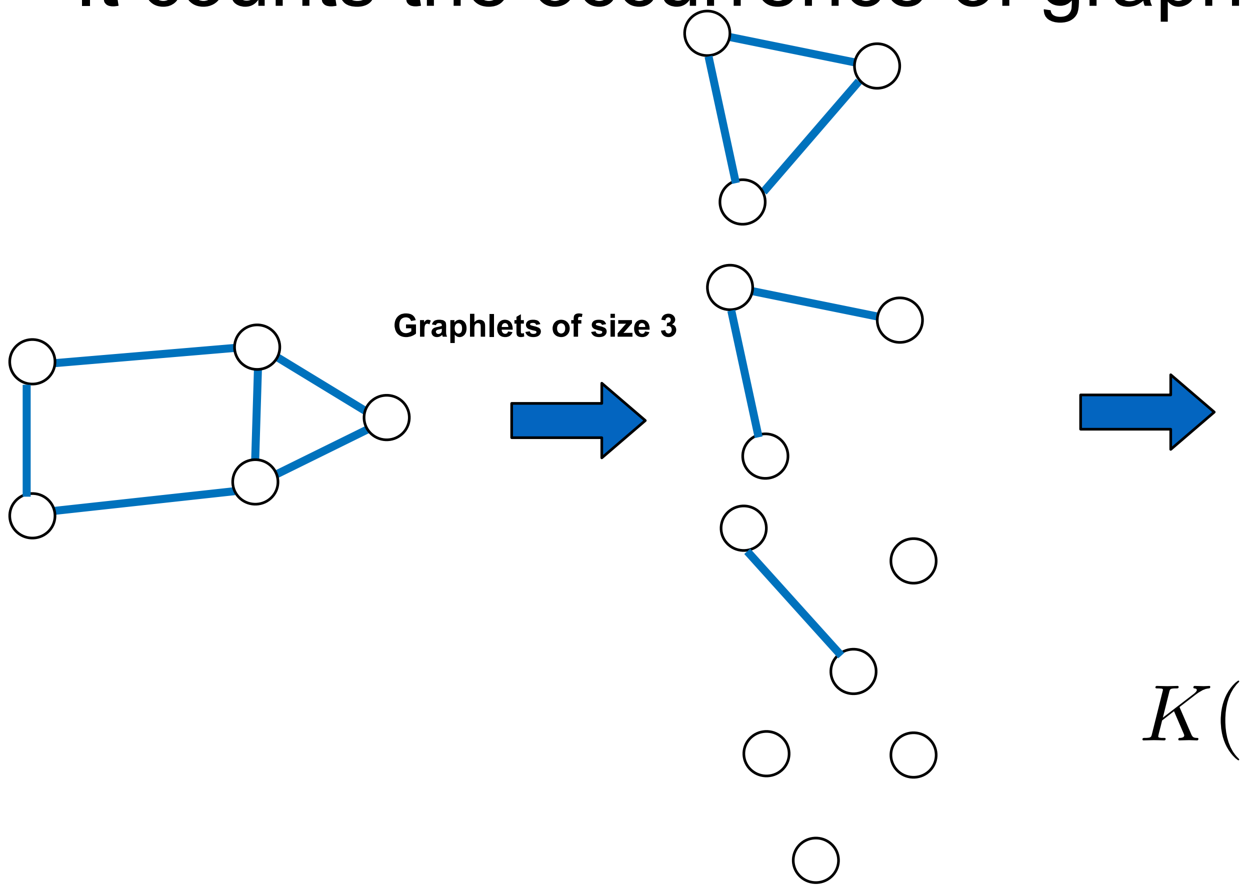


$\phi(\mathcal{G}_1) = [0, 3, 2, 0, 1]$ $\neq$ $\phi(\mathcal{G}_2) = [0, 5, 1, 0, 1]$

- Limitation: It can miss global properties of the graph

Graphlet kernel

- A subgraph-based kernel based on graphlets
  - Nodes do not need to be connected
- It counts the occurrence of graphlets in a graph



$$\phi(\mathcal{G}) = [1, 6, 3, 0]$$

Graphlet kernel

$$K(\mathcal{G}_1, \mathcal{G}_2) = < \phi(\mathcal{G}_1), \phi(\mathcal{G}_2) >$$

- Limitation: High complexity; there are $\binom{N}{K}$ graphlets of K nodes

[Shervashidze et al., Efficient graphite kernels for large graph comparison. AISTATS, 2009]

Weisfeiler-Lehman kernel

- Iteratively aggregate information from node's neighbourhoods wider than the 1-hop neighborhood

- **Color refinement algorithm:**
  - **Input:** a graph $\mathcal{G}$
  - Assign an initial color $c^{(0)}(u)$ (e.g., node degree) to each node u of $\mathcal{G}$
  - For each iteration $k+1$ refine node colors as

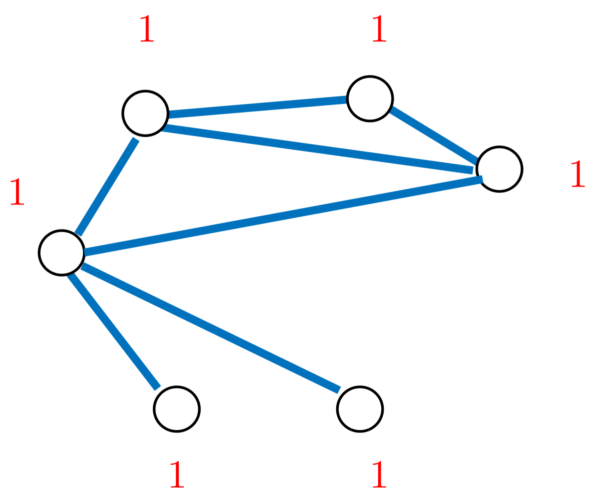
$$c^{(k+1)}(u) = \mathrm{HASH}\left(\left\{c^{(k)}(u), \{c^{(k)}(v)\}_{v \in \mathcal{N}_v}\right\}\right)$$

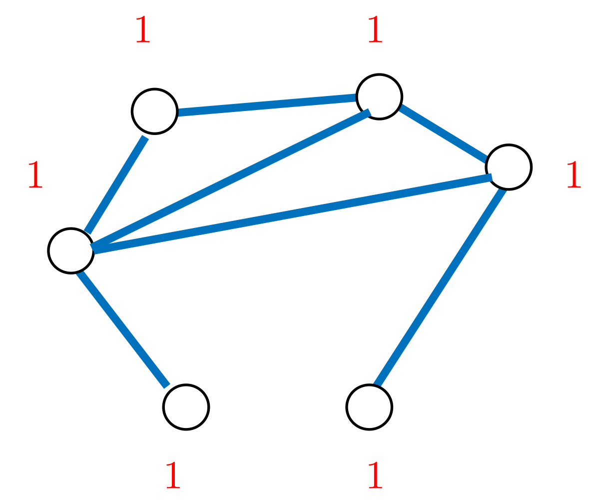
  - Output: The node color $c^{(K)}(u)$ after K iterations

- It provides a description of the K-hop neighborhood with an efficient algorithm

[Shervashidze et al., Weisfeiler-Lehman graph kernels, JMLR, 2011]

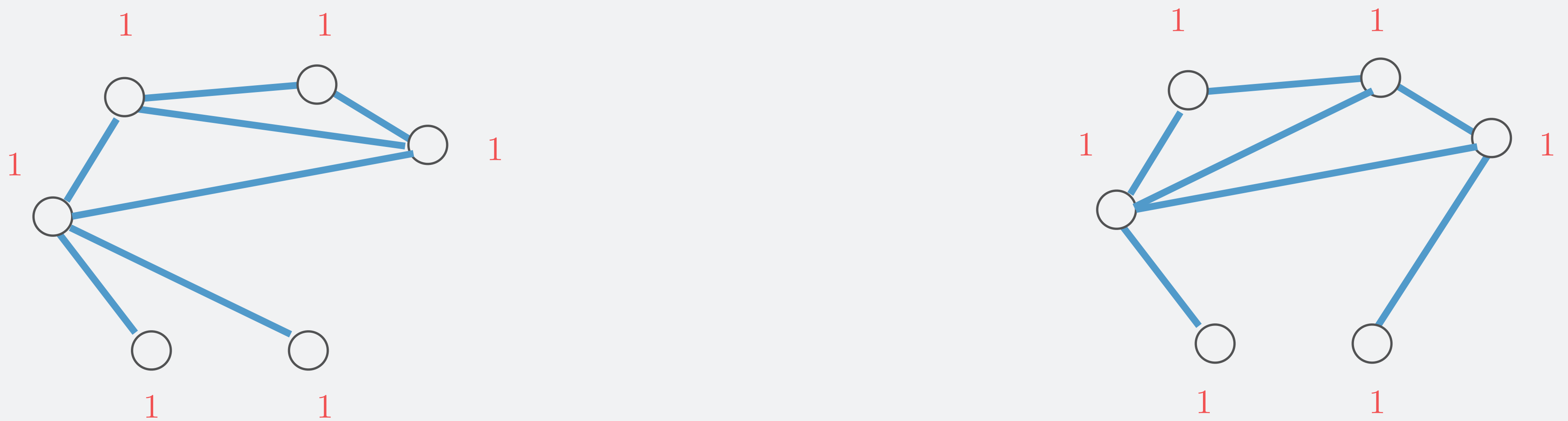
Example of WL kernel



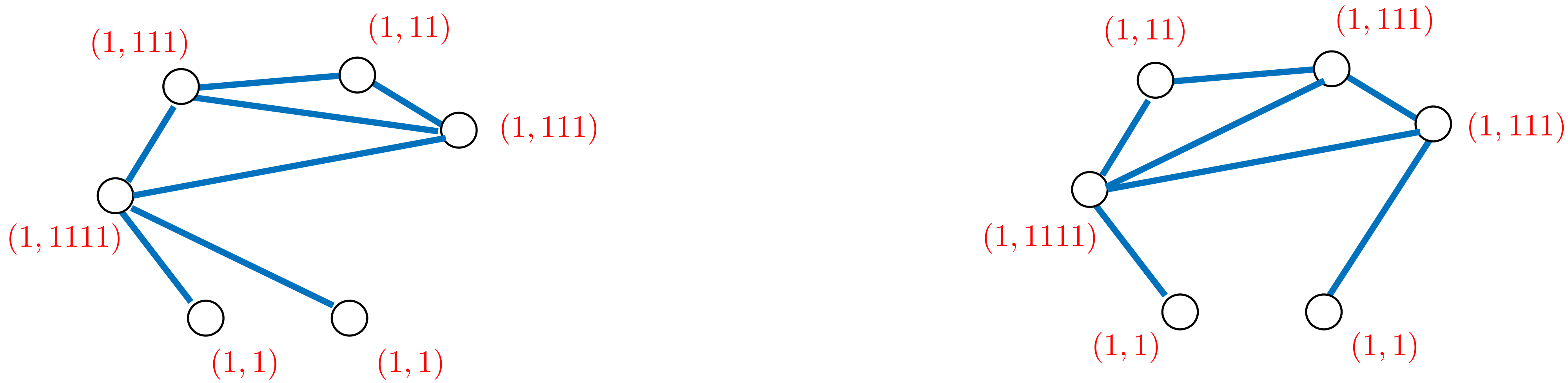


Initial colors

Example of WL kernel

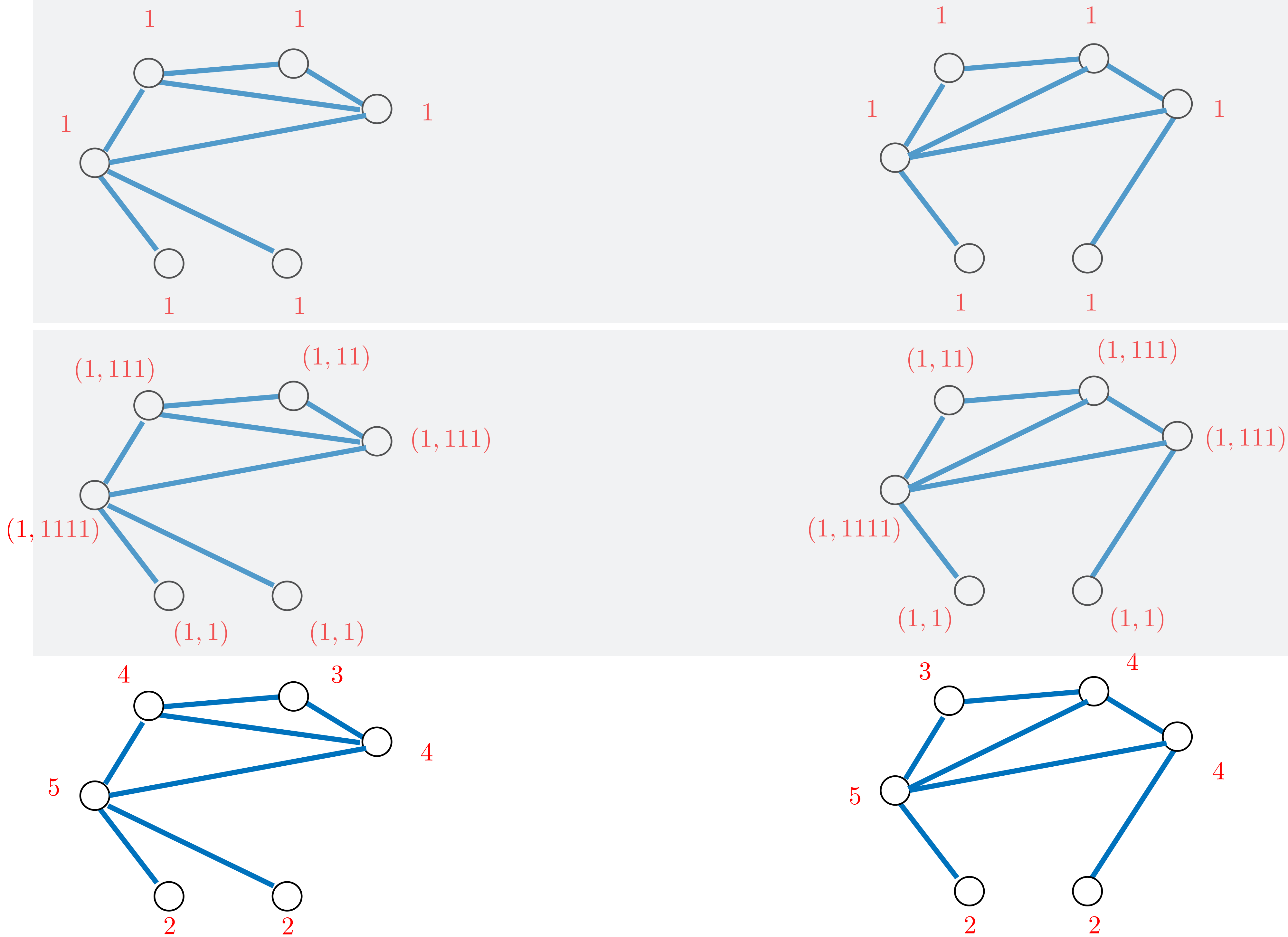


Initial colors



Aggregate neighbourhood colors

Example of WL kernel



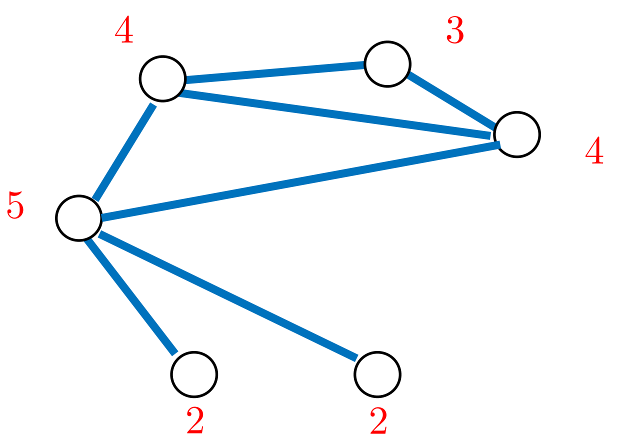
Initial colors

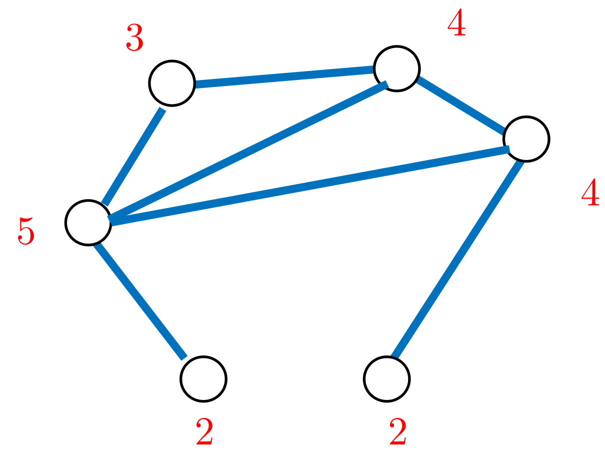
Aggregate neighbourhood colors

Hash table

$(1,1)$	$\rightarrow$	2
$(1,11)$	$\rightarrow$	3
$(1,111)$	$\rightarrow$	4
$(1,1111)$	$\rightarrow$	5

Example of WL kernel

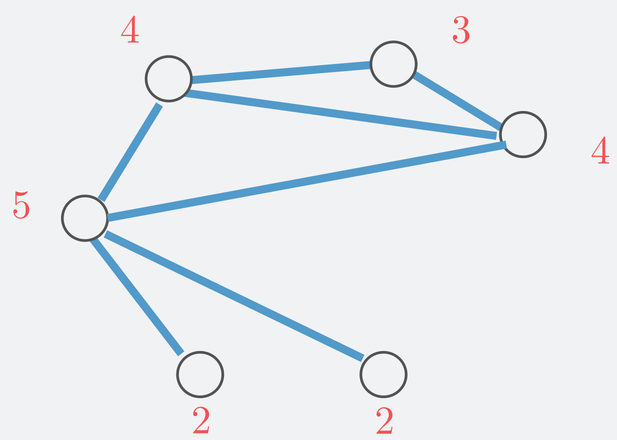


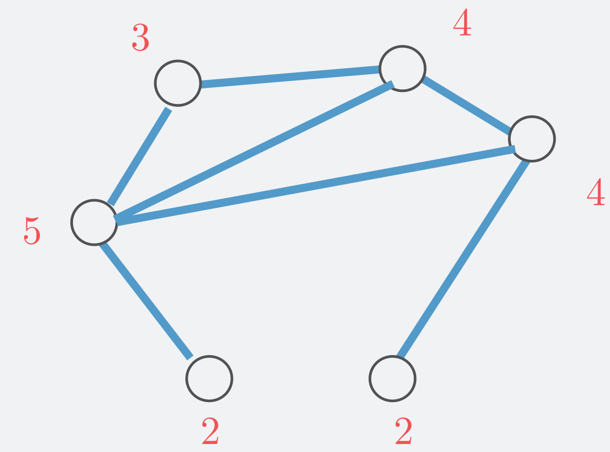


Hash table

$(1,1)$	$\rightarrow$	2
$(1,11)$	$\rightarrow$	3
$(1,111)$	$\rightarrow$	4
$(1,1111)$	$\rightarrow$	5

Example of WL kernel





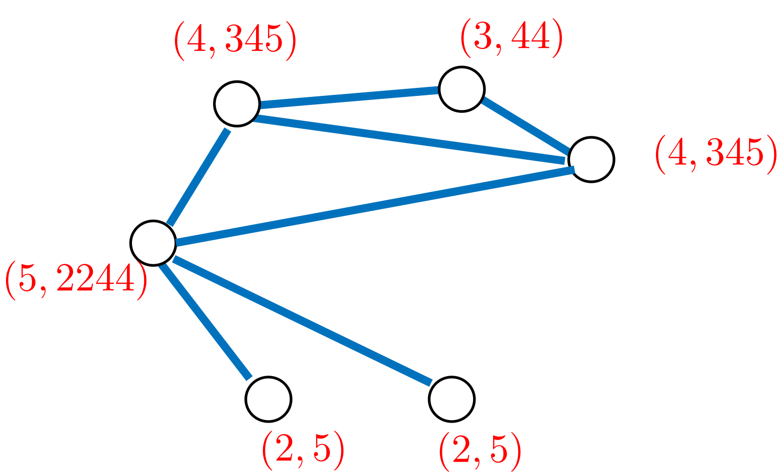
Hash table

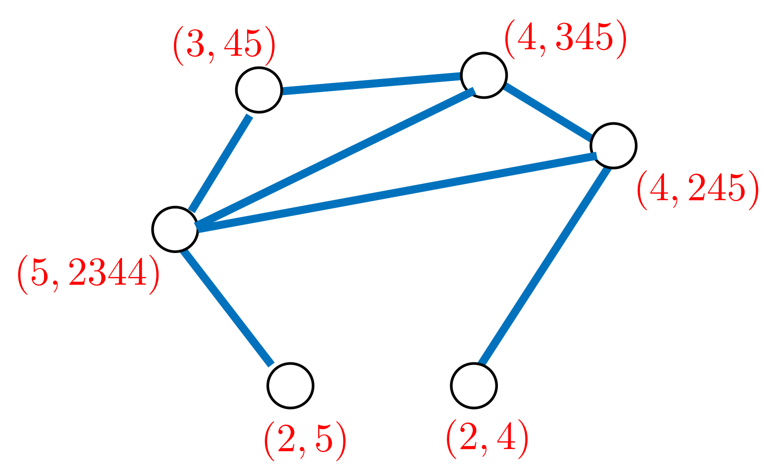
$(1,1) \rightarrow 2$

$(1,11) \rightarrow 3$

$(1,111) \rightarrow 4$

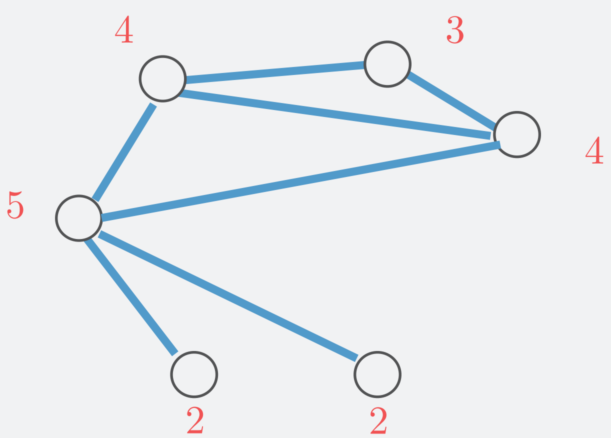
$(1,1111) \rightarrow 5$

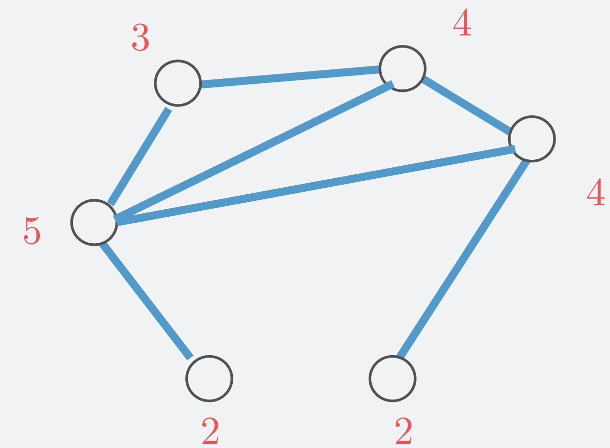




Aggregate neighbourhood colors

Example of WL kernel





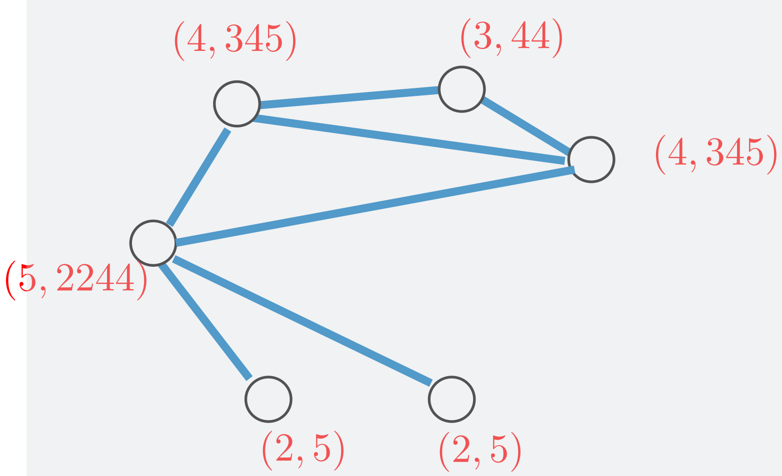
Hash table

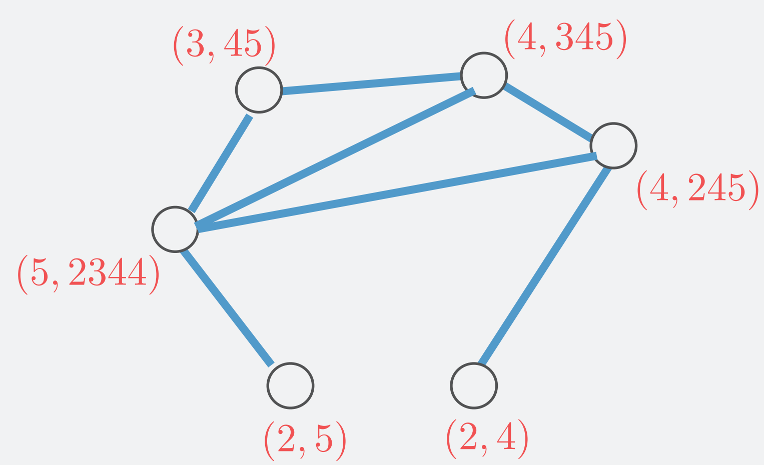
$(1,1) \rightarrow 2$

$(1,11) \rightarrow 3$

$(1,111) \rightarrow 4$

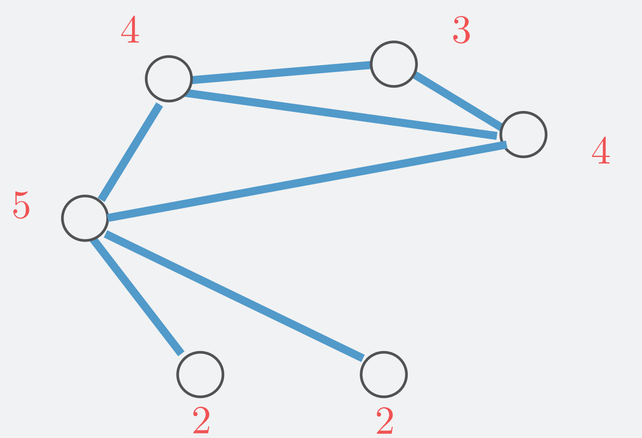
$(1,1111) \rightarrow 5$

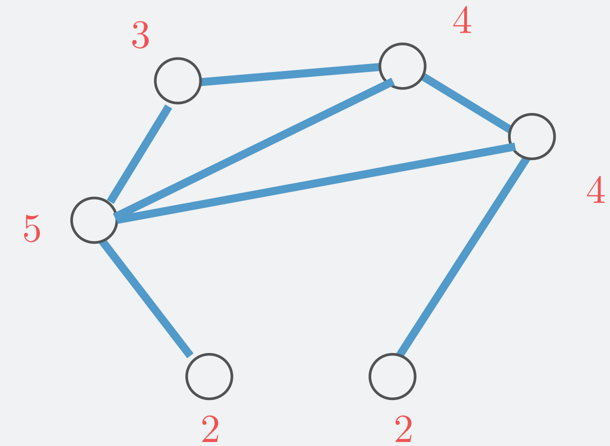




Aggregate neighbourhood colors

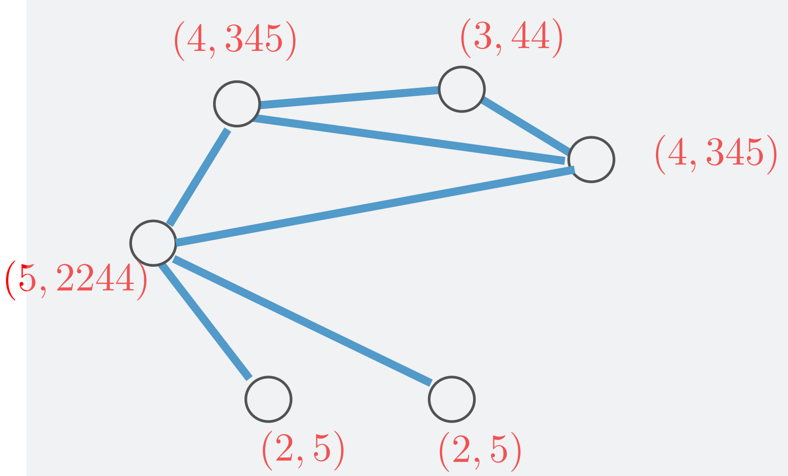
Example of WL kernel

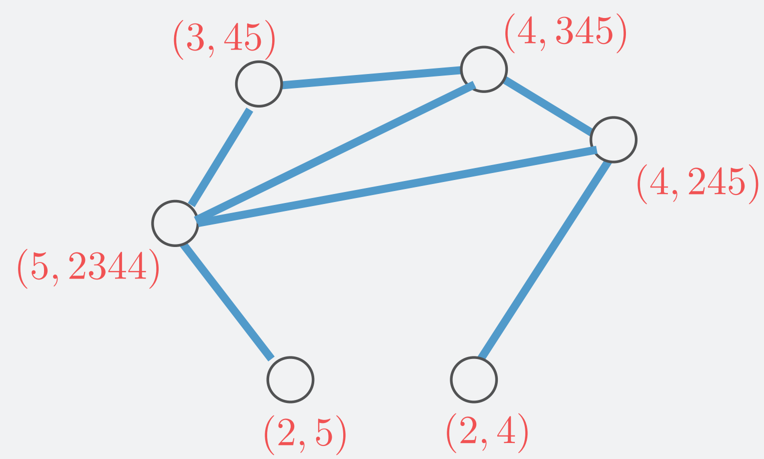




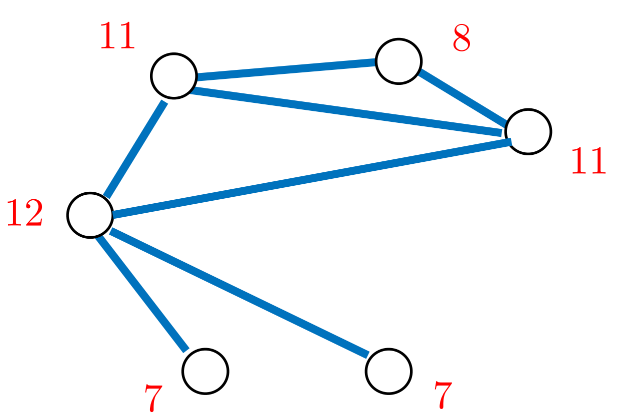
Hash table

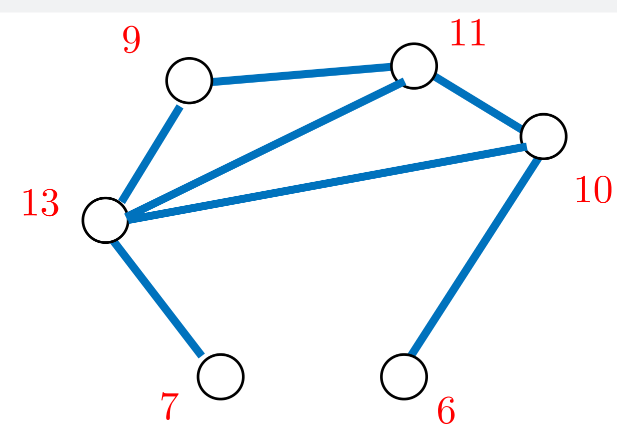
$(1,1)$	$\rightarrow$	2
$(1,11)$	$\rightarrow$	3
$(1,111)$	$\rightarrow$	4
$(1,1111)$	$\rightarrow$	5





Aggregate neighbourhood colors



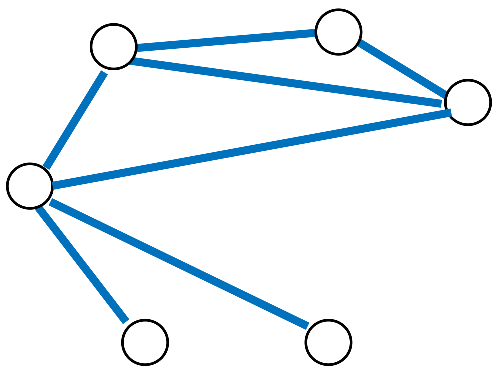


Hash table

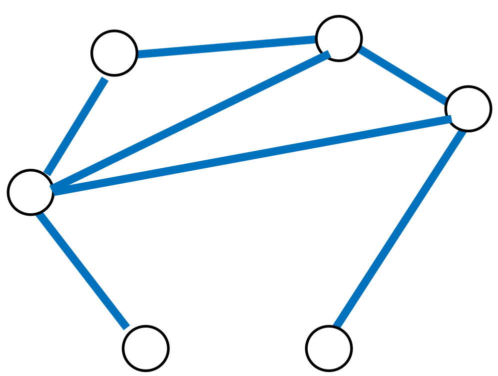
$(2,4)$	$\rightarrow$	6
$(2,5)$	$\rightarrow$	7
$(3,44)$	$\rightarrow$	8
$(3,45)$	$\rightarrow$	9
$(4,245)$	$\rightarrow$	10
$(4,345)$	$\rightarrow$	11
$(5,2244)$	$\rightarrow$	12
$(5,2344)$	$\rightarrow$	13

Example of WL kernel

- After K iterations, the WL kernel computes the histogram of colors



$$\begin{array}{l} \phantom{\phi(\mathcal{G}_1) = [}{\color{red}1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13} \\ \phi(\mathcal{G}_1) = [6, 2, 1, 2, 1, 0, 2, 1, 0, 0, 2, 1, 0] \end{array}$$

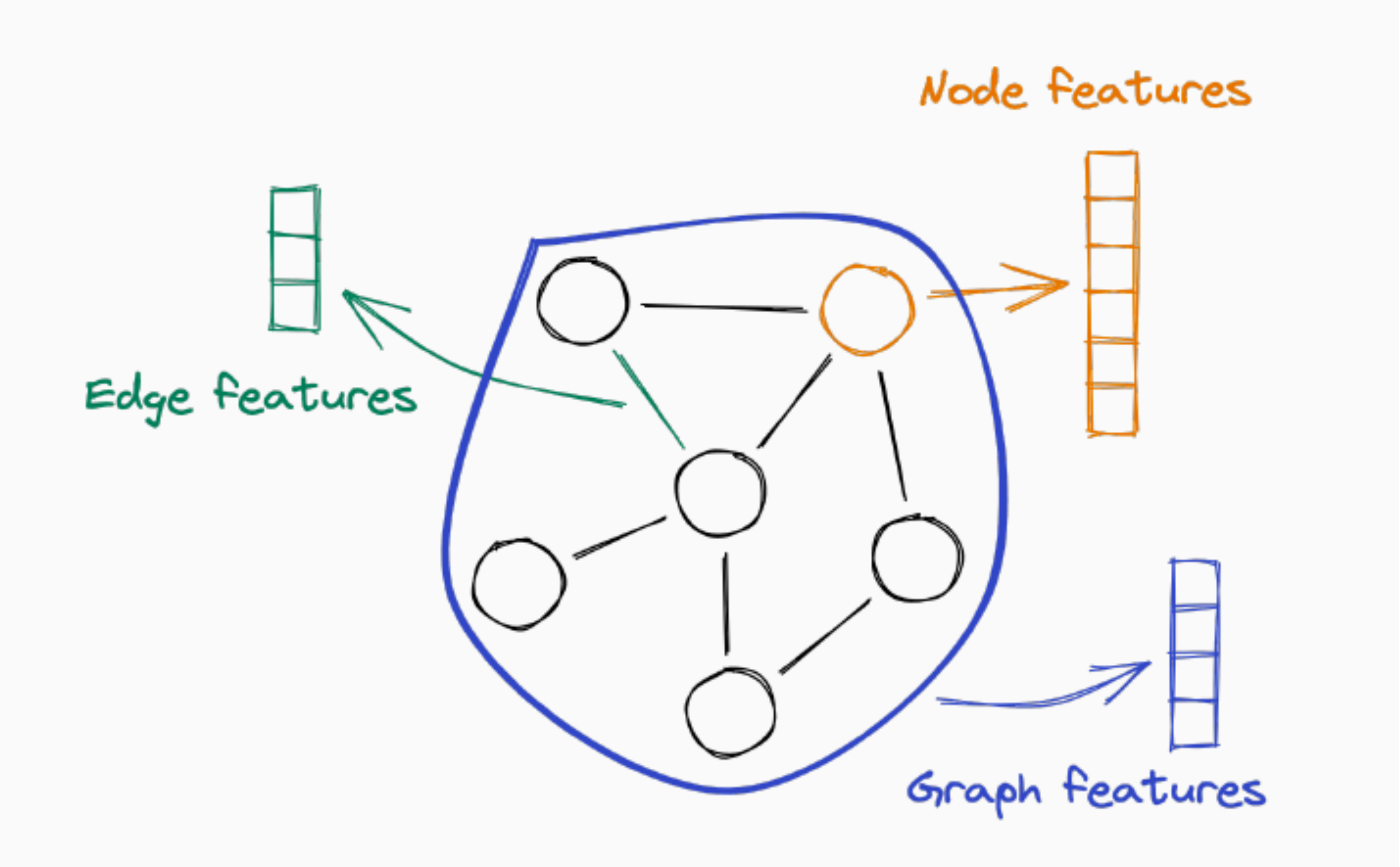


$$\begin{array}{l} \phantom{\phi(\mathcal{G}_2) = [}{\color{red}1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13} \\ \phi(\mathcal{G}_2) = [6, 2, 1, 2, 1, 1, 1, 0, 1, 1, 1, 0, 1] \end{array}$$

WL kernel

$$K(\mathcal{G}_1, \mathcal{G}_2) =< \phi(\mathcal{G}_1), \phi(\mathcal{G}_2) >$$

Edge level features

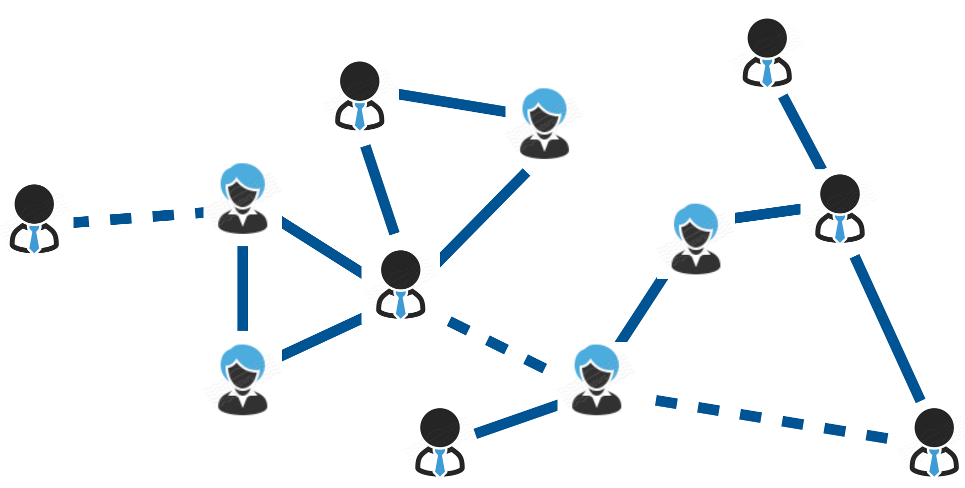


How can we capture relationships between neighboring nodes?

Illustrative example: Link prediction

- Goal: Predict the existence of an edge between two nodes given some already existing edges

- Intuition: Design features about pairs of nodes that measure the overlap between their neighborhoods



Link prediction in one slide

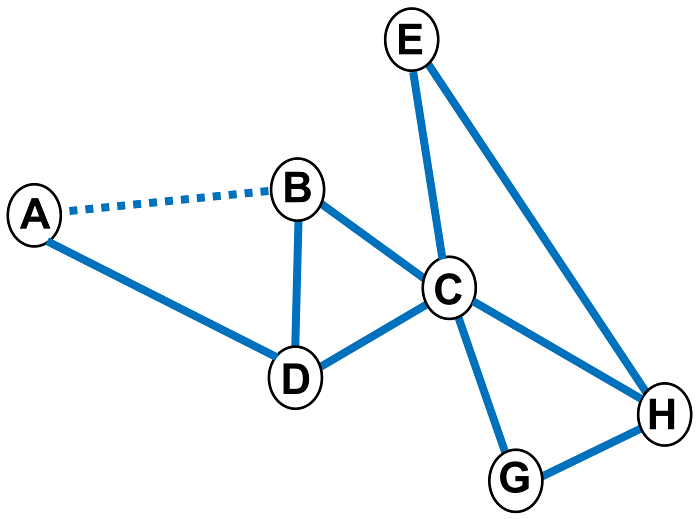
- For each pair of nodes (u, v) compute $score(u, v)$
- Sort pairs by decreasing score
- Predict top k pairs as a link
- Common ways to compute $score(\cdot, \cdot)$:
  - Local neighborhood overlap: quantify the similarity of the neighborhood between two nodes
  - Global neighborhood overlap: quantify if two nodes belong to the same community in the graph

Local neighborhood overlap

- Compute $score(u, v)$ as the number of common neighboring nodes i.e., **overlap** between (u, v)

- **Common neighbors:**

$$score(A, B) = |\mathcal{N}_A \cap \mathcal{N}_B| = |\{D\}| = 1$$



- **Jaccard's coefficient:**

$$score(A, B) = \frac{|\mathcal{N}_A \cap \mathcal{N}_B|}{|\mathcal{N}_A \cup \mathcal{N}_B|} = \frac{|\{D\}|}{|\{D, C\}|} = \frac{1}{2}$$

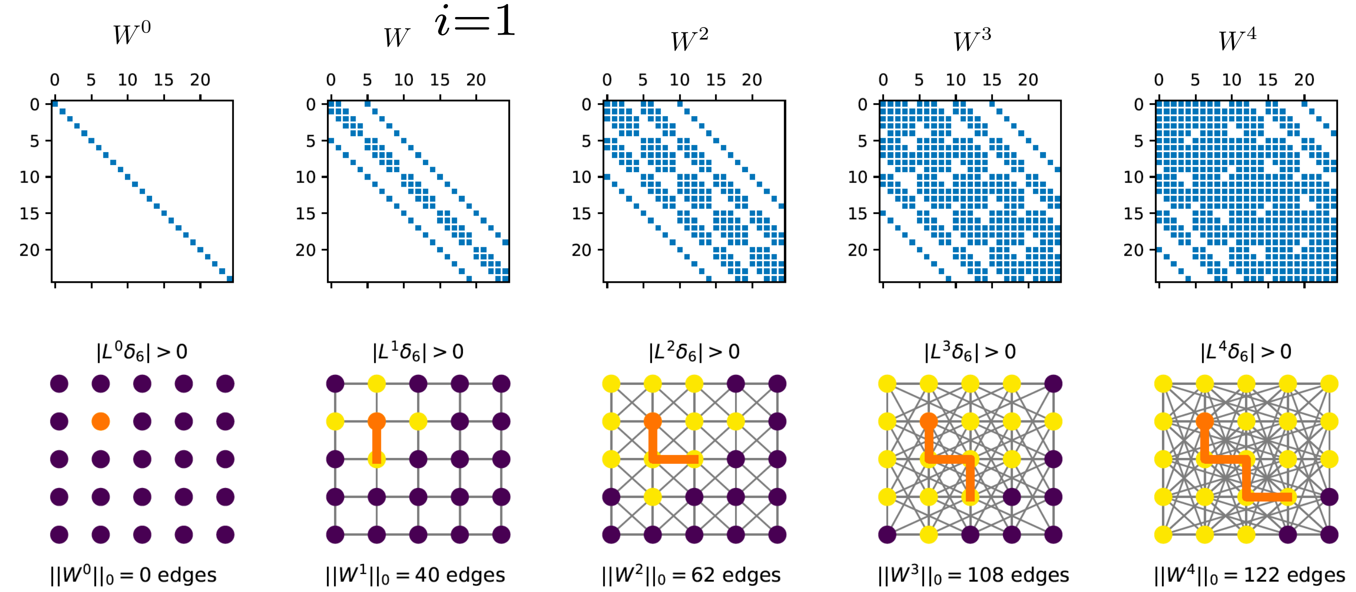
- **Limitation:** A link between (A, E) cannot be created. Why?

Global neighborhood overlap

- Compute $score(u, v)$ by taking into consideration the entire graph, i.e., global overlap

- **Katz index:** Count the number of walks of all lengths between a given pair of nodes
  - Use powers of the adjacency/weight matrix

$$score(A, B) = \sum_{i=1}^{\infty} \beta^i W_{AB}^i = [(I - \beta W)^{-1} - I]_{AB}$$



Summary

- Traditional graph analysis/inference pipeline decouples the data representation and learning process
  - Hand-crafted features + ML/statistics

- The type of features depends on the task:
  - Node level: generate features for each individual node
    - Node degree, centrality, clustering coefficients, graphlets
  - Graph level: generate features for the whole graph
    - Bag of nodes, graphlet kernels, WL kernels
  - Link level: generate features that measure a common neighborhood between two nodes
    - Local/global neighborhood overlap

- Careful design of graph features can be useful in applications where data is limited

Limitations

- Hand-engineered features are defined a priori: no adaptation to the data
- Designing graph features can very often be a time consuming and expensive process
- Not easy to incorporate additional features on the nodes
- More flexibility can be achieved with an end-to-end learning pipeline: next lectures!

References

1. Graph representation learning (chap 2), William Hamilton

2. Metrics for graph comparison: A practitioner's guide, Wills, PLOS ONE, 2020
   - https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0228728

3. Graph Kernels State-of-the-Art and Future Challenges, Borgwardt et al,
   - https://arxiv.org/pdf/2011.03854.pdf