Modelling and Analysis of Complex Networks

— Semester 3, Master of Data Science —

Jun Pang
University of Luxembourg

Machine Learning on Graphs

Machine Learning

Why learning?

- There is no need to 'learn' to calculate payroll.
- Learning is used when:
  - human expertise does not exist (bioinformatics)
  - humans are unable to explain their expertise (speech recognition, computer vision)
  - solutions change in time and need adapting to new cases
- Learning: modifying a behaviour based on experience

Machine Learning

Why learning?

- There is no need to 'learn' to calculate payroll.
- Learning is used when:
  - human expertise does not exist (bioinformatics)
  - humans are unable to explain their expertise (speech recognition, computer vision)
  - solutions change in time and need adapting to new cases
- Learning: modifying a behaviour based on experience

Machine Learning

How are things learned?

- Memorisation
  - accumulate individual facts
  - limited by time to observe facts and memory to store facts
- Generalisation
  - deduce new facts from old facts
  - limited by the accuracy of the deduction process
- Machine learning: make programs that can infer useful information from implicit patterns in data

Machine Learning

How are things learned?

- Memorisation
  - accumulate individual facts
  - limited by time to observe facts and memory to store facts
- Generalisation
  - deduce new facts from old facts
  - limited by the accuracy of the deduction process
- Machine learning: make programs that can infer useful information from implicit patterns in data

Machine Learning

What is machine learning?

- Observe a set of examples (training data)
- Infer something about the process that generated the data (model)
- Use inference to make predictions about previously unseen data (test data)

Programming computers to learn

$$f_\theta(x)$$

Machine Learning

- Supervised machine learning: given training examples in the form of inputs labelled with corresponding outputs
  - Classification: the output domain is discrete
  - Regression: the output domain is continuous
- Unsupervised machine learning: given unlabelled inputs
  - Clustering: discovering structure
  - Dimensionality reduction: reducing the number of variables
- Reinforcement learning: data in the form of sequences of actions, observations, and rewards; the goal is to produce a policy for acting

Machine Learning

- Supervised machine learning: given training examples in the form of inputs labelled with corresponding outputs
  - Classification: the output domain is discrete
  - Regression: the output domain is continuous
- Unsupervised machine learning: given unlabelled inputs
  - Clustering: discovering structure
  - Dimensionality reduction: reducing the number of variables
- Reinforcement learning: data in the form of sequences of actions, observations, and rewards; the goal is to produce a policy for acting

Machine Learning

- Supervised machine learning: given training examples in the form of inputs labelled with corresponding outputs
  - Classification: the output domain is discrete
  - Regression: the output domain is continuous
- Unsupervised machine learning: given unlabelled inputs
  - Clustering: discovering structure
  - Dimensionality reduction: reducing the number of variables
- Reinforcement learning: data in the form of sequences of actions, observations, and rewards; the goal is to produce a policy for acting

Machine Learning

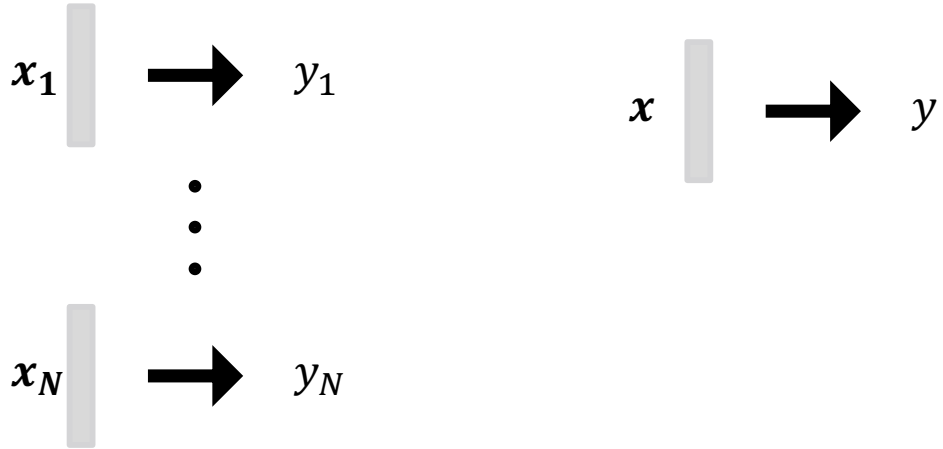
Traditional (supervised) ML pipeline:

- Train an ML model
  - Random forest, SVM, neural networks, etc.
- Apply the model
  - Given a new node/link/graph, obtain its features, and make a prediction

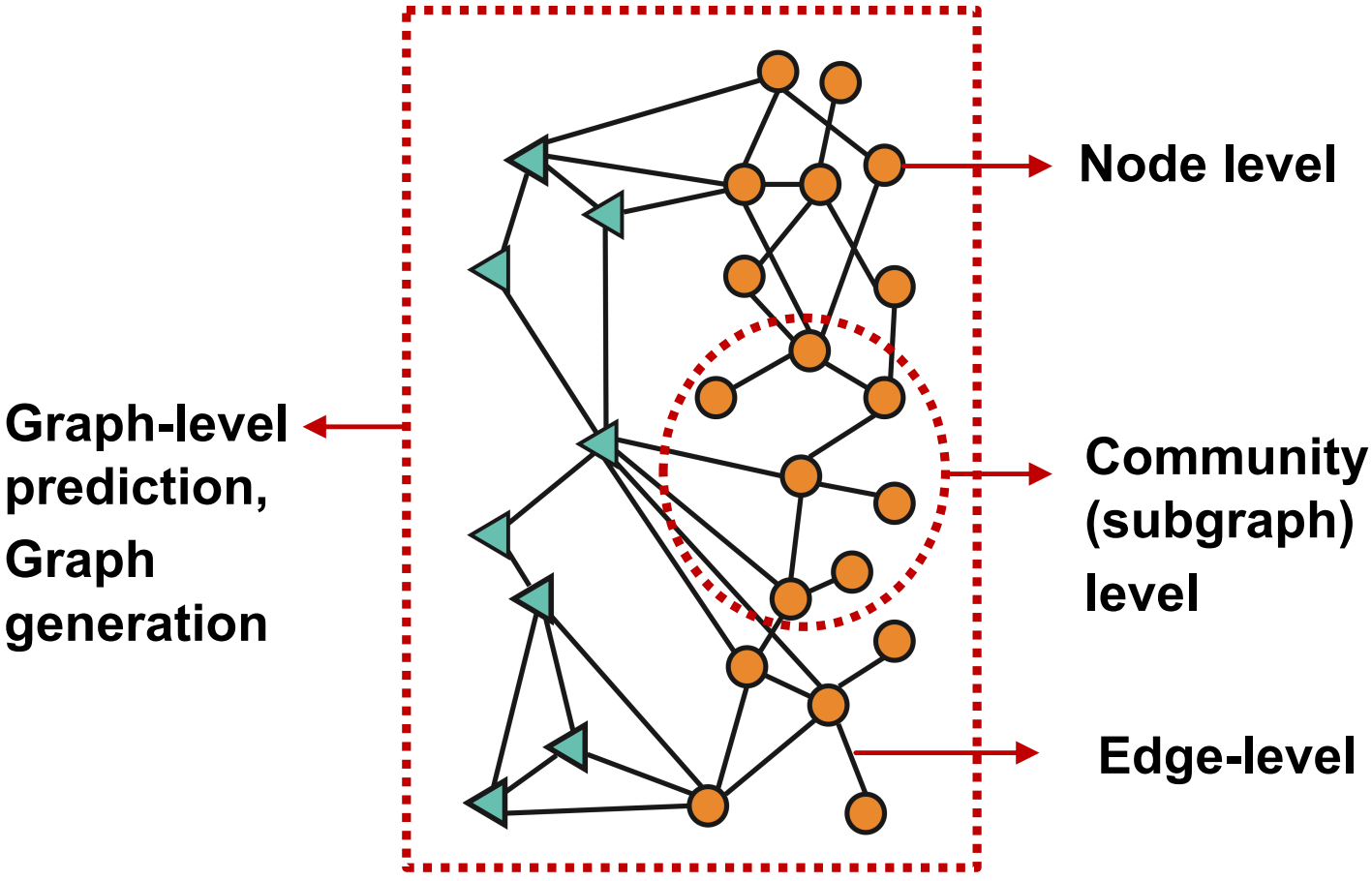
Machine Learning

Traditional (supervised) ML pipeline:

- Train an ML model
  - Random forest, SVM, neural networks, etc.
- Apply the model
  - Given a new node/link/graph, obtain its features, and make a prediction



Machine Learning on Graphs



Machine Learning on Graphs

Feature design:

- Using effective features over graphs is the key to achieving good test performance
- Traditional ML pipeline uses hand-designed features
- We overview the traditional features for:
  - Node-level prediction
  - Link-level prediction
  - Graph-level prediction
- For simplicity, we focus on undirected graphs.

Machine Learning on Graphs

Feature design:

- Using effective features over graphs is the key to achieving good test performance
- Traditional ML pipeline uses hand-designed features
- We overview the traditional features for:
  - Node-level prediction
  - Link-level prediction
  - Graph-level prediction
- For simplicity, we focus on undirected graphs.

Machine Learning on Graphs

Design choices: for making predictions for a set of objects

- Features: d-dimensional vectors
- Objects: nodes, edges, sets of nodes, entire graphs
- Objective function: depending on what tasks we aim to solve

Machine learning on graphs:

Given $G = (V, E)$, learn a function $f : V \to \mathbb{R}^d$.

Machine Learning on Graphs

Design choices: for making predictions for a set of objects

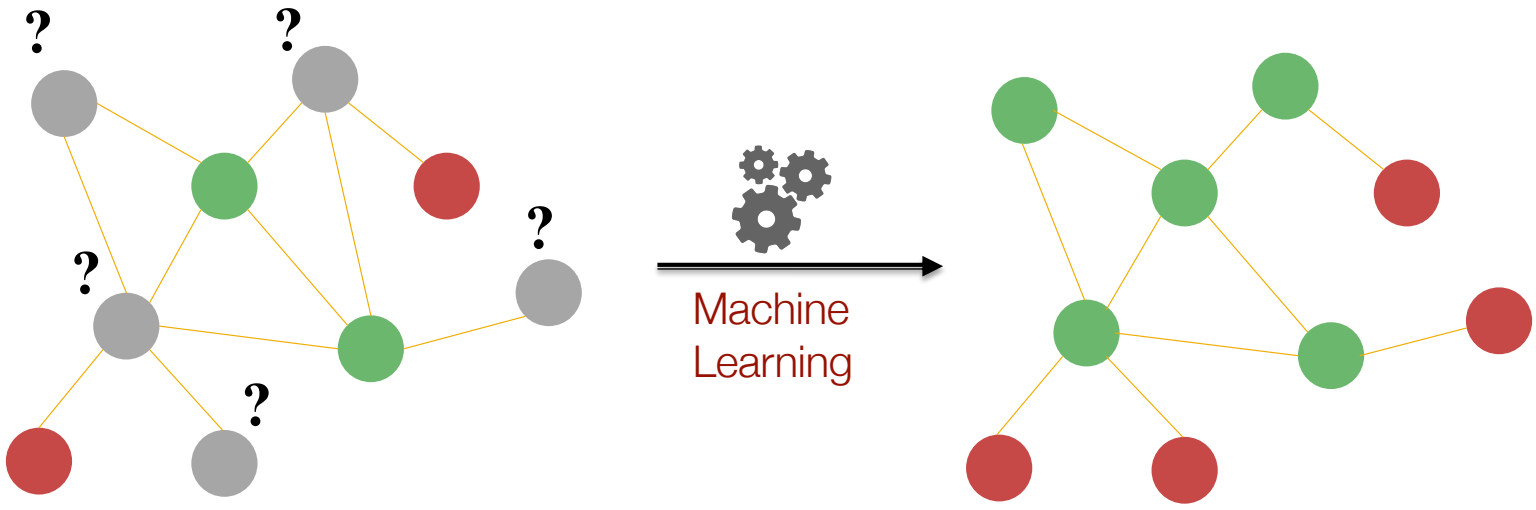
- Features: d-dimensional vectors
- Objects: nodes, edges, sets of nodes, entire graphs
- Objective function: depending on what tasks we aim to solve

Machine learning on graphs:

Given $G = (V, E)$, learn a function $f : V \to \mathbb{R}^d$.

Node Classification

Node Classification



Node classification: ML requires features!

Node Classification

Node-level features characterise the structure and position of a node in the network.

- Node degree counts the neighbouring nodes without capturing their importance
- Node centrality takes the node importance in a graph into account (betweenness centrality, closeness centrality, etc.)
- Clustering coefficient counts the number of triangles in the ego-network
- Graphlets: generalise 'triangles' to 'pre-specified' subgraphs (graphlet degree vector counts the number of graphlets that a node touches)

Node Classification

Node-level features characterise the structure and position of a node in the network.

- Node degree counts the neighbouring nodes without capturing their importance
- Node centrality takes the node importance in a graph into account (betweenness centrality, closeness centrality, etc.)
- Clustering coefficient counts the number of triangles in the ego-network
- Graphlets: generalise 'triangles' to 'pre-specified' subgraphs (graphlet degree vector counts the number of graphlets that a node touches)

Node Classification

We have introduced different ways to obtain node features, and they can be categorised as:

- Importance-based features: useful for predicting influential nodes in a graph
  - Node degree, different node centrality measures
- Structure-based features: useful for predicting a particular role a node plays in a graph
  - Node degree, clustering coefficient, graphlet count vector

Node Classification

We have introduced different ways to obtain node features, and they can be categorised as:

- Importance-based features: useful for predicting influential nodes in a graph
  - Node degree, different node centrality measures
- Structure-based features: useful for predicting a particular role a node plays in a graph
  - Node degree, clustering coefficient, graphlet count vector

Link Prediction

Link Prediction

- Do you know why Facebook "People you may know" is so accurate?
- How Youtube/Spotify/Amazon recommend you the right items?
- Link prediction! (More generally, recommendation. But link prediction is a popular way to do it.)

Link Prediction

- Do you know why Facebook "People you may know" is so accurate?
- How Youtube/Spotify/Amazon recommend you the right items?
- Link prediction! (More generally, recommendation. But link prediction is a popular way to do it.)

Link Prediction

- Do you know why Facebook "People you may know" is so accurate?
- How Youtube/Spotify/Amazon recommend you the right items?
- Link prediction! (More generally, recommendation. But link prediction is a popular way to do it.)

Link Prediction

- Observed network: current state
- Link prediction: what edges
  - might appear in the future (future link prediction)
  - might have been missed (missing link prediction)
- Link prediction based on network structure (local vs. global)
- Link prediction based on node properties & network structure

The key is to design features for a pair of nodes!

Link Prediction

- Observed network: current state
- Link prediction: what edges
  - might appear in the future (future link prediction)
  - might have been missed (missing link prediction)
- Link prediction based on network structure (local vs. global)
- Link prediction based on node properties & network structure

The key is to design features for a pair of nodes!

Link Prediction

- Observed network: current state
- Link prediction: what edges
  - might appear in the future (future link prediction)
  - might have been missed (missing link prediction)
- Link prediction based on network structure (local vs. global)
- Link prediction based on node properties & network structure

The key is to design features for a pair of nodes!

Link Prediction

- Observed network: current state
- Link prediction: what edges
  - might appear in the future (future link prediction)
  - might have been missed (missing link prediction)
- Link prediction based on network structure (local vs. global)
- Link prediction based on node properties & network structure

The key is to design features for a pair of nodes!

Link Prediction

Link prediction via proximity:

- For each pair of nodes (x, y), compute score $c(x, y)$
  - For example, $c(x, y)$ could be the number of common neighbours 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

Link Prediction

Link prediction via proximity:

- For each pair of nodes (x, y), compute score $c(x, y)$
  - For example, $c(x, y)$ could be the number of common neighbours 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

Link Prediction

Link prediction via proximity: different scoring functions $c(x, y)$

- Graph distance: (negated) shortest path length
- Common neighbours: $| N_i \cap N_j |$
- Jaccard's coefficient: $\frac{|N_i \cap N_j|}{|N_i| + |N_j| - |N_i \cap N_j|}$
- Adamic and Adar score: $\sum_{k \in N_i \cap N_j} \frac{1}{|N_k|}$
- PageRank: $r_x(y) + r_y(x)$, $r_x(y)$ is the stationary distribution score of y under the random walk: with probability 0.15, jump to x; with 0.85, go to a random neighbour of the current node

Link Prediction

Link prediction via proximity: different scoring functions $c(x, y)$

- Graph distance: (negated) shortest path length
- Common neighbours: $| N_i \cap N_j |$
- Jaccard's coefficient: $\frac{|N_i \cap N_j|}{|N_i| + |N_j| - |N_i \cap N_j|}$
- Adamic and Adar score: $\sum_{k \in N_i \cap N_j} \frac{1}{|N_k|}$
- PageRank: $r_x(y) + r_y(x)$, $r_x(y)$ is the stationary distribution score of y under the random walk: with probability 0.15, jump to x; with 0.85, go to a random neighbour of the current node

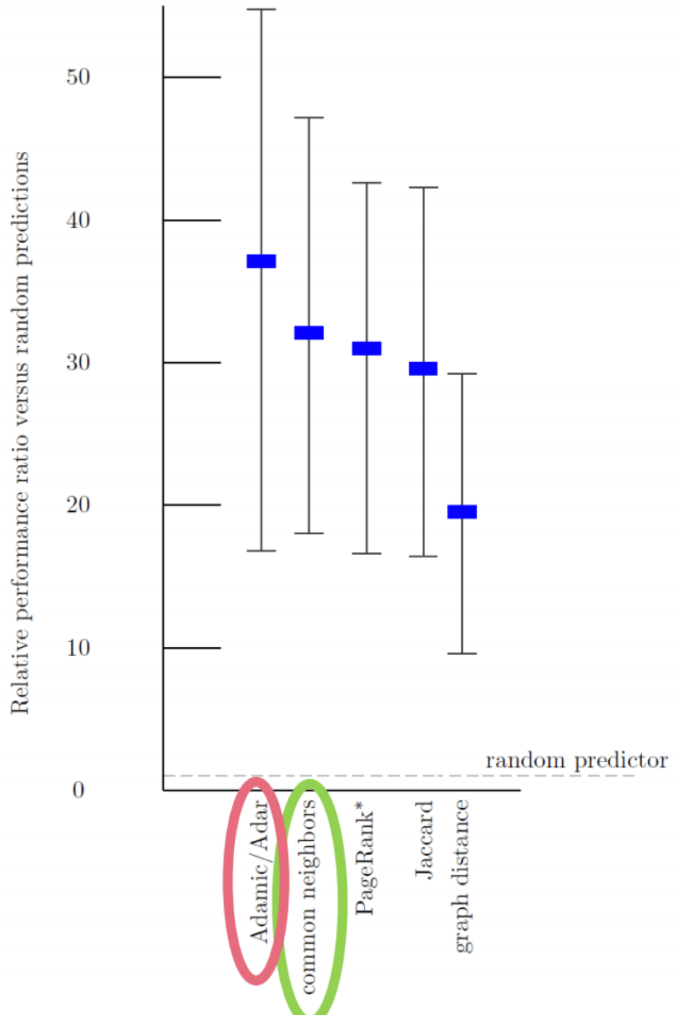
Link Prediction

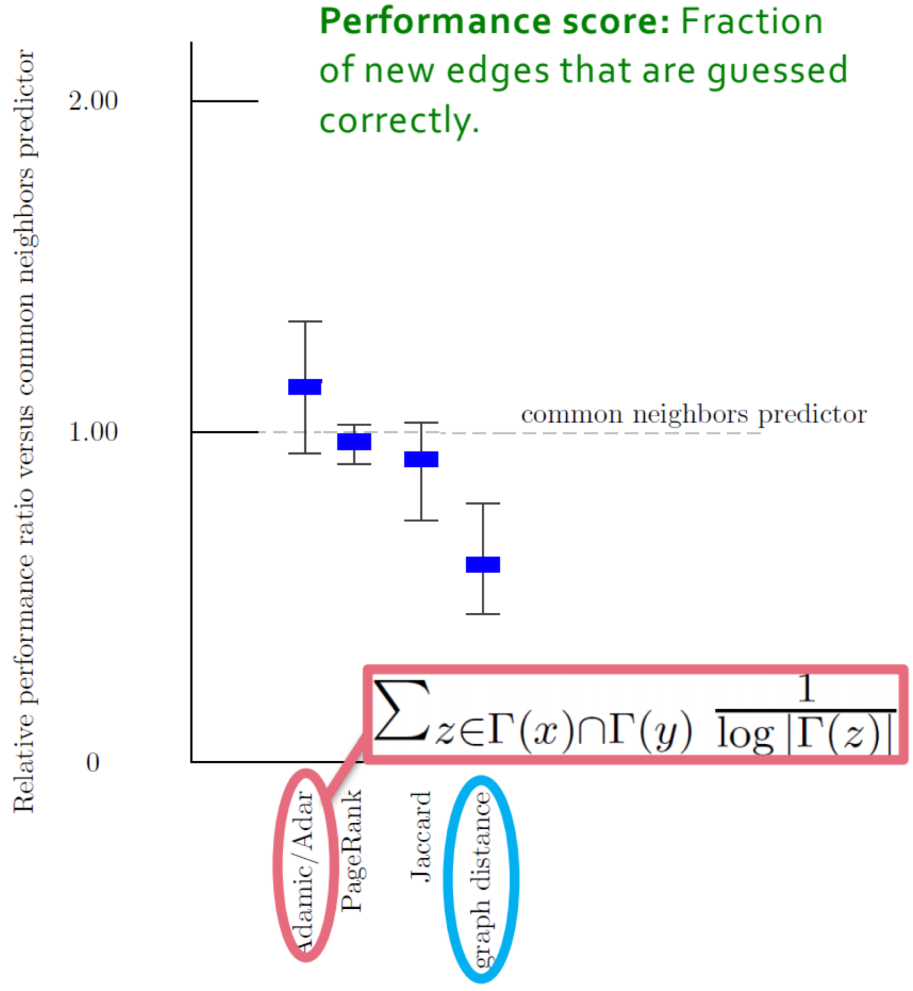
Link prediction via proximity: different scoring functions $c(x, y)$

- Graph distance: (negated) shortest path length
- Common neighbours: $| N_i \cap N_j |$
- Jaccard's coefficient: $\frac{|N_i \cap N_j|}{|N_i| + |N_j| - |N_i \cap N_j|}$
- Adamic and Adar score: $\sum_{k \in N_i \cap N_j} \frac{1}{|N_k|}$
- PageRank: $r_x(y) + r_y(x)$, $r_x(y)$ is the stationary distribution score of y under the random walk: with probability 0.15, jump to x; with 0.85, go to a random neighbour of the current node

Link Prediction

Link prediction via proximity: evaluation results





[LibenNowell-Kleinbverg'03]

Link Prediction

Link prediction via network structure:

- Compute community structure on the whole graph
- Assign a high score for two nodes in the same community, a low score otherwise

How to choose the score?

Link Prediction

Link prediction via network structure:

- Compute community structure on the whole graph
- Assign a high score for two nodes in the same community, a low score otherwise

How to choose the score?

Link Prediction

Link prediction via network structure: How to choose the score?

- For methods based on a quality function optimisation (e.g., modularity), assign a score to each pair proportional to the change in quality function association with adding an edge between them.
- For example, Louvain algorithm optimises modularity:
  - Each edge added between communities decreases in the modularity
  - Each edge added inside community increases in the modularity

Link Prediction

Link prediction via network structure: How to choose the score?

- For methods based on a quality function optimisation (e.g., modularity), assign a score to each pair proportional to the change in quality function association with adding an edge between them.
- For example, Louvain algorithm optimises modularity:
  - Each edge added between communities decreases in the modularity
  - Each edge added inside community increases in the modularity

Link Prediction

- Distance-based features: use the shortest path length between two nodes but do not capture how neighbourhood overlaps.
  - Graph distance
- Local neighbourhood overlap: captures how many neighbouring nodes are shared by two nodes.
  - Common neighbours, Jaccard's coefficient, Adamic and Adar score
- Global neighbourhood overlap: uses global graph structure to score two nodes.
  - PageRank, community structure

Link Prediction

- Distance-based features: use the shortest path length between two nodes but do not capture how neighbourhood overlaps.
  - Graph distance
- Local neighbourhood overlap: captures how many neighbouring nodes are shared by two nodes.
  - Common neighbours, Jaccard's coefficient, Adamic and Adar score
- Global neighbourhood overlap: uses global graph structure to score two nodes.
  - PageRank, community structure

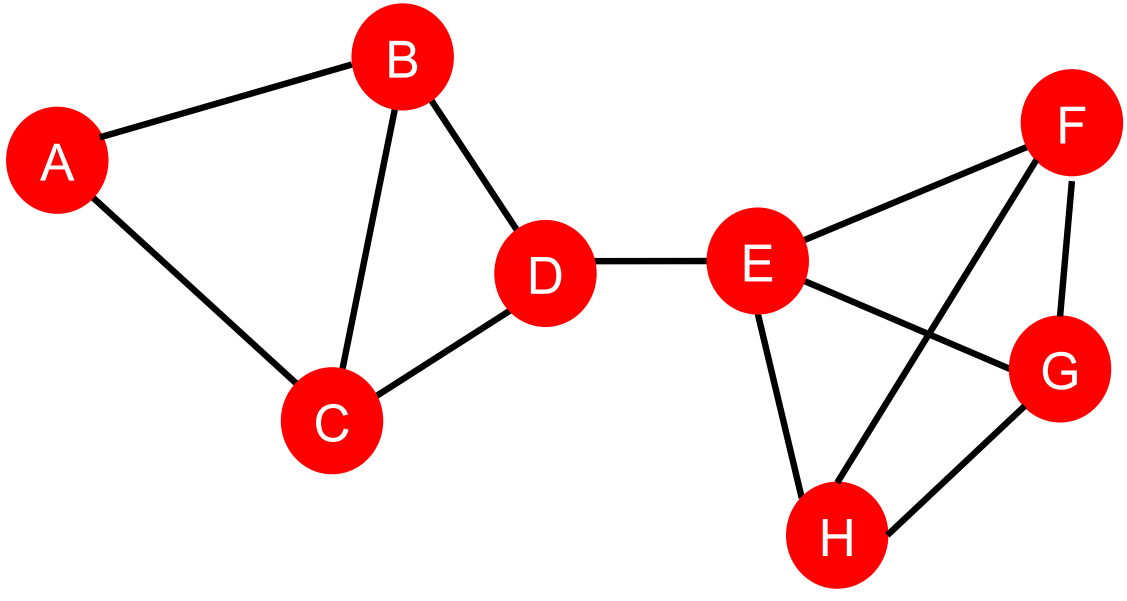
Link Prediction

- Distance-based features: use the shortest path length between two nodes but do not capture how neighbourhood overlaps.
  - Graph distance
- Local neighbourhood overlap: captures how many neighbouring nodes are shared by two nodes.
  - Common neighbours, Jaccard's coefficient, Adamic and Adar score
- Global neighbourhood overlap: uses global graph structure to score two nodes.
  - PageRank, community structure

Graph-Level Features

Graph-Level Features

We want features that characterise the structure of an entire graph.



Graph-Level Features

Kernel methods are widely-used for traditional ML for graph-level prediction.

- Kernel $K(G, G') \in \mathbb{R}$ measures similarity between data
- There exists a feature representation $\phi(\cdot)$, such that $K(G, G') = \phi(G)^T \phi(G')$
- Once the kernel is defined, an off-the-shelf ML model, such as kernel SVM, can be used to make predictions.

Graph-Level Features

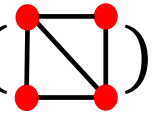
Graph kernels measure similarity between two graphs.

- Graphlet kernel
- Weisfeiler-Lehman kernel
- Others: random-walk kernel, shortest-path graph kernel, etc.

Graph Kernel

Design graph feature vector $\phi(G)$: bag-of-words (BoW) for a graph

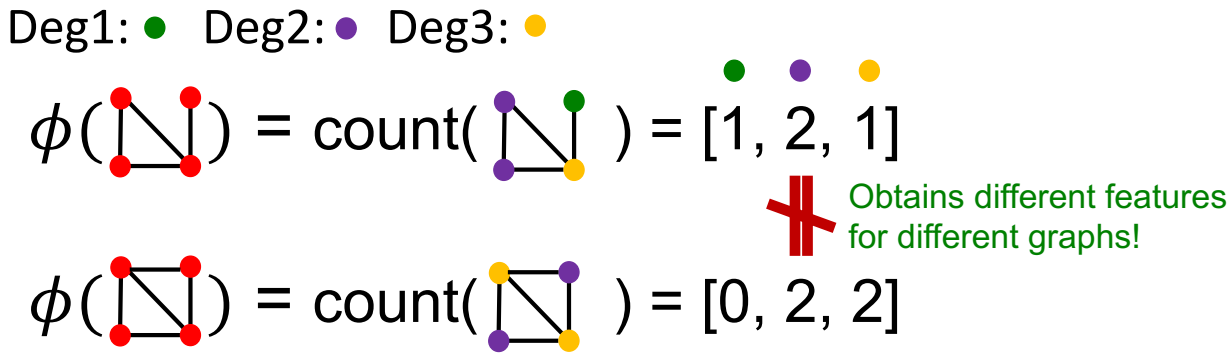
- BoW simply uses the word counts as features for documents (no ordering considered)
- Naïve extension to graphs: regard nodes as words

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

- Since both graphs have 4 red nodes, we get the same feature vector for two different graphs.

Graph Kernel

- What if we use bag-of-node degrees?



- Both Graphlet kernel and Weisfeiler-Lehman (WL) kernel use bag-of-$*$ representation of graph, where $*$ is more sophisticated than node degrees.

Graphlet Kernel

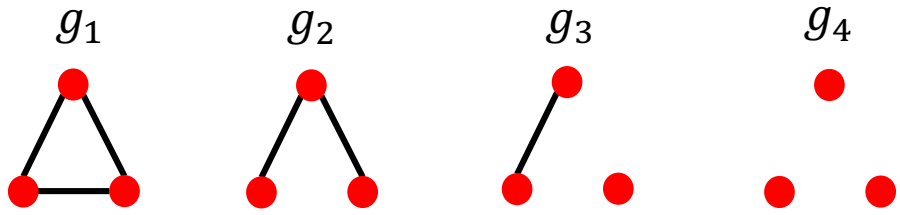
Graphlet features: count the number of different graphlets.

- The definition of graphlets here is slightly different from node-level features.
- Nodes in graphlets here do not need to be connected (allows for isolated nodes).
- The graphlets here are not rooted.

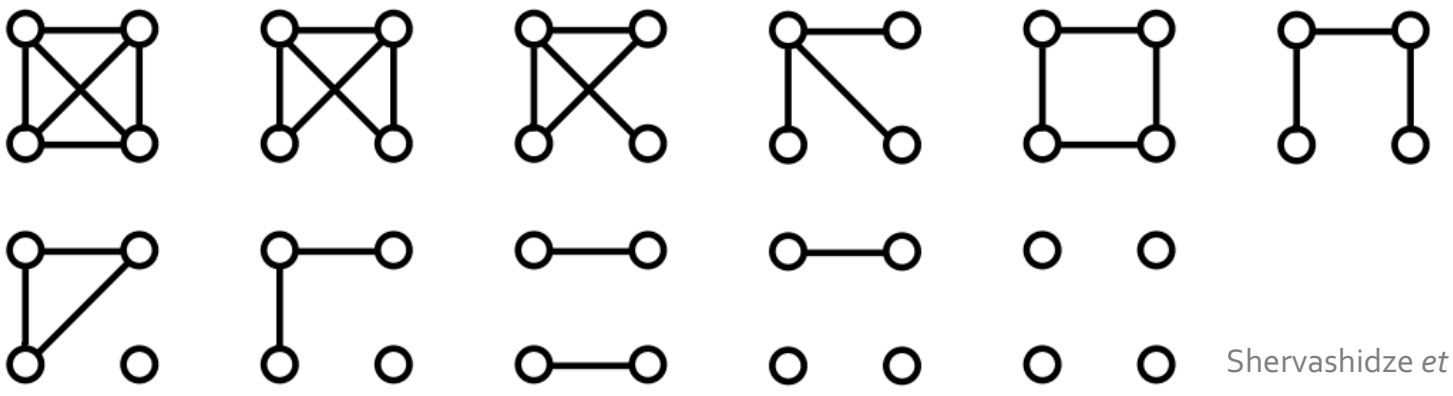
Graphlet Kernel

Graphlet features: $G_k = (g_1, \ldots, g_{n_k})$, a list of graphlets of size k.

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



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



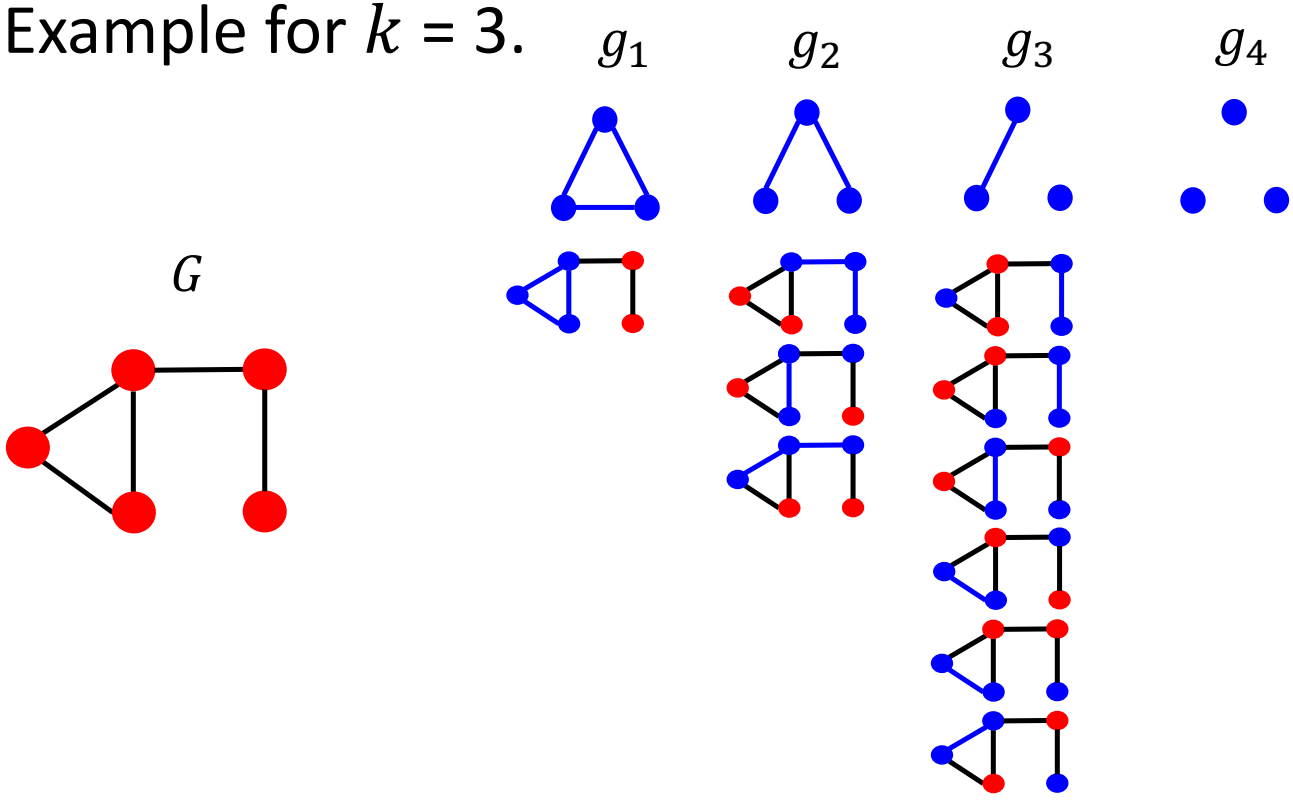
Shervashidze *et al.*, AISTATS 2011

Graphlet Kernel

Given a graph G, and a graphlet list $G_k = (g_1, g_2, \ldots, g_{n_k})$, define the graphlet count vector $f_G \in \mathbb{R}^{n_k}$ as

$$(f_G)_i = \#(g_i \subset G) \text{ for } i = 1, 2, \ldots, n_k$$

- Example for $k = 3$.



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

Graphlet Kernel

Given two graphs G and G', graph kernel is computed as

$$K(G, G') = f_G^T f_{G'}$$

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

$$h(G) = \frac{f_g}{\sum(f_G)} \qquad K(G, G') = h_G^T h_{G'}$$

- Limitation: Counting graphlets is expensive!

  (If a graph's node degree is bounded by d, $\mathcal{O}(nd^{k-1})$ algorithm exists to count all the graphlets of size k.)

Graphlet Kernel

Given two graphs G and G', graph kernel is computed as

$$K(G, G') = f_G^T f_{G'}$$

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

$$h(G) = \frac{f_g}{\sum(f_G)} \qquad K(G, G') = h_G^T h_{G'}$$

- Limitation: Counting graphlets is expensive!

  (If a graph's node degree is bounded by d, $\mathcal{O}(nd^{k-1})$ algorithm exists to count all the graphlets of size k.)

Graphlet Kernel

Given two graphs G and G', graph kernel is computed as

$$K(G, G') = f_G^T f_{G'}$$

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

$$h(G) = \frac{f_g}{\sum(f_G)} \qquad K(G, G') = h_G^T h_{G'}$$

- Limitation: Counting graphlets is expensive!

  (If a graph's node degree is bounded by d, $\mathcal{O}(nd^{k-1})$ algorithm exists to count all the graphlets of size k.)

Graphlet Kernel

Given two graphs G and G', graph kernel is computed as

$$K(G, G') = f_G^T f_{G'}$$

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

$$h(G) = \frac{f_g}{\sum(f_G)} \qquad K(G, G') = h_G^T h_{G'}$$

- Limitation: Counting graphlets is expensive!

  (If a graph's node degree is bounded by d, $\mathcal{O}(nd^{k-1})$ algorithm exists to count all the graphlets of size k.)

Weisfeiler-Lehman (WL) Kernel

How to design an efficient graph feature description $\phi(G)$?

- Use neighbourhood structure to iteratively enrich node vocabulary
- Generalised version of bag-of-node degrees since node degrees are one-hop neighbourhood information.
- Algorithm to achieve this: colour refinement

Weisfeiler-Lehman (WL) Kernel

Colour refinement: given a graph G with a set of nodes V

- Assign an initial colour $c^{(0)}(v)$ to each node v
- Iteratively refine node colours by

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

  where **HASH** maps different inputs to different colours.
- After K steps of colour refinement, $c^{(K)}(v)$ summarises the structure of K-hop neighbourhood.

Weisfeiler-Lehman (WL) Kernel

Colour refinement: given a graph G with a set of nodes V

- Assign an initial colour $c^{(0)}(v)$ to each node v
- Iteratively refine node colours by

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

  where **HASH** maps different inputs to different colours.
- After K steps of colour refinement, $c^{(K)}(v)$ summarises the structure of K-hop neighbourhood.

Weisfeiler-Lehman (WL) Kernel

Colour refinement: given a graph G with a set of nodes V

- Assign an initial colour $c^{(0)}(v)$ to each node v
- Iteratively refine node colours by

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

  where **HASH** maps different inputs to different colours.

- After K steps of colour refinement, $c^{(K)}(v)$ summarises the structure of K-hop neighbourhood.

Weisfeiler-Lehman (WL) Kernel

Colour refinement: given a graph G with a set of nodes V

- Assign an initial colour $c^{(0)}(v)$ to each node v
- Iteratively refine node colours by

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

  where **HASH** maps different inputs to different colours.
- After K steps of colour refinement, $c^{(K)}(v)$ summarises the structure of K-hop neighbourhood.

Weisfeiler-Lehman (WL) Kernel

Colour refinement: an example, given two graphs

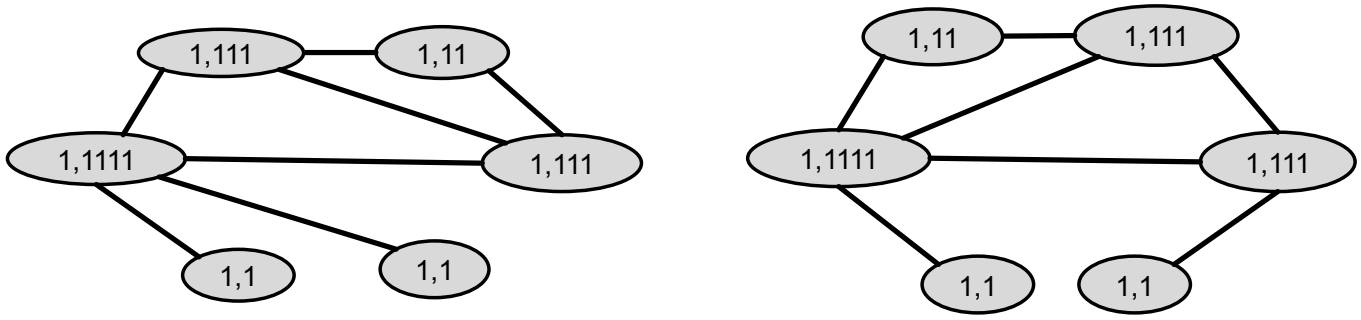
- Assign initial colours ($k = 0$)

Weisfeiler-Lehman (WL) Kernel

Colour refinement: an example, given two graphs

- Assign initial colours ($k = 0$)

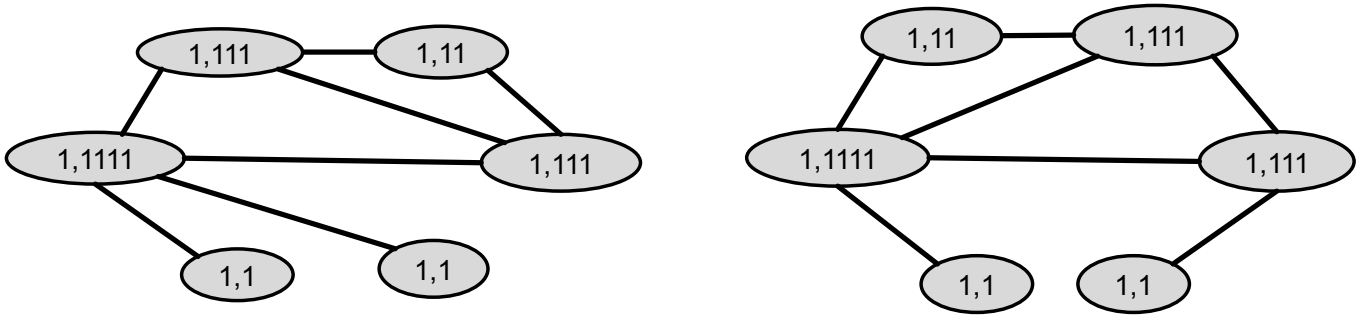
- Aggregate neighbouring colours



Weisfeiler-Lehman (WL) Kernel

Colour refinement: an example, given two graphs

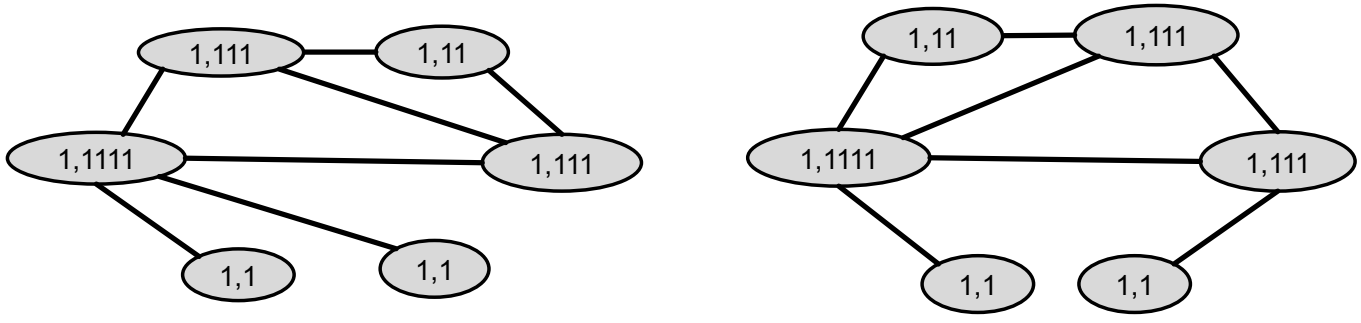
- Aggregated colours



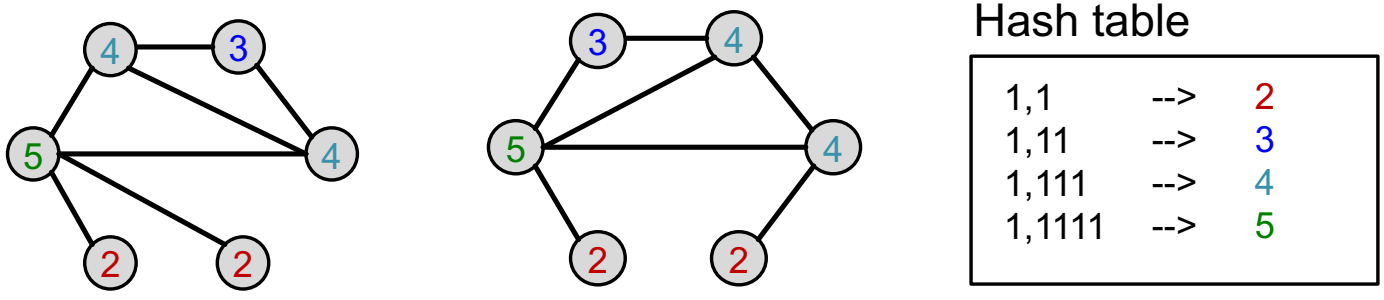
Weisfeiler-Lehman (WL) Kernel

Colour refinement: an example, given two graphs

- Aggregated colours



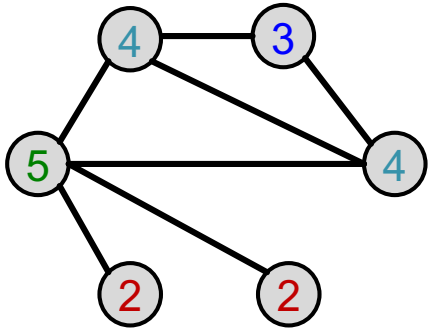
- Hash aggregated colours ($k = 1$)

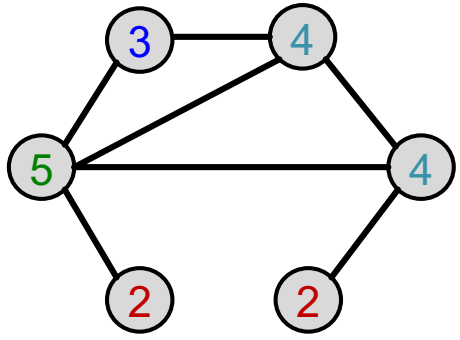


Weisfeiler-Lehman (WL) Kernel

Colour refinement: an example, given two graphs

- 1st iteration colours ($k = 1$)

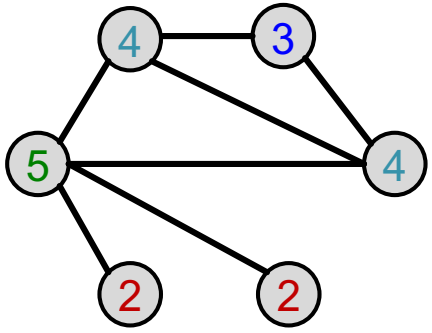


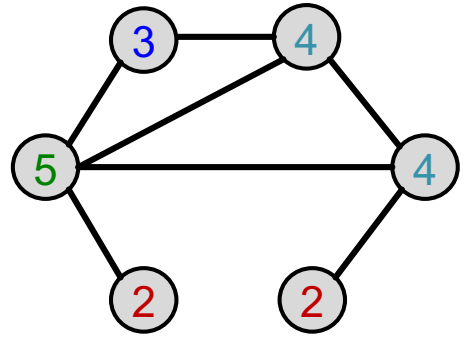


Weisfeiler-Lehman (WL) Kernel

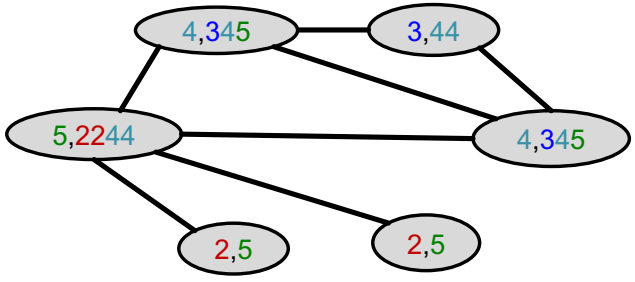
Colour refinement: an example, given two graphs

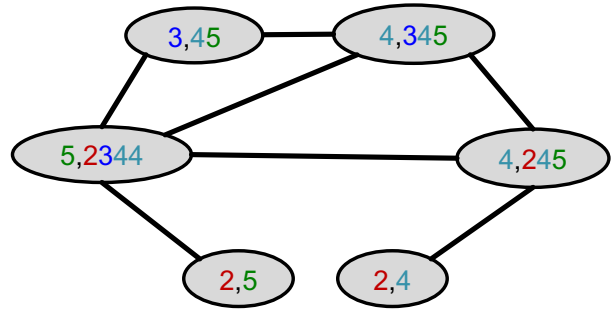
- 1st iteration colours ($k = 1$)





- Aggregate neighbouring colours

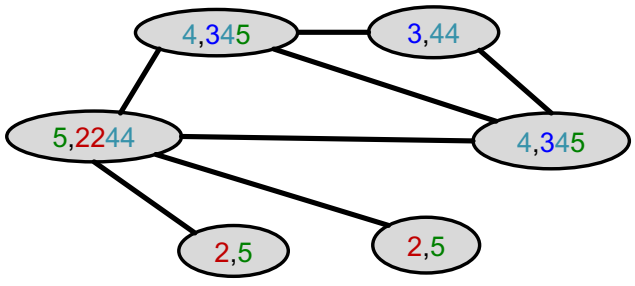


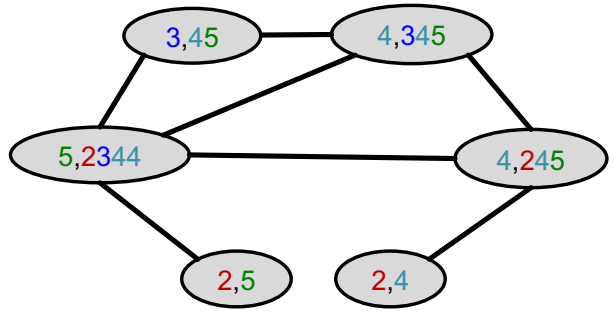


Weisfeiler-Lehman (WL) Kernel

Colour refinement: an example, given two graphs

- Aggregated colours

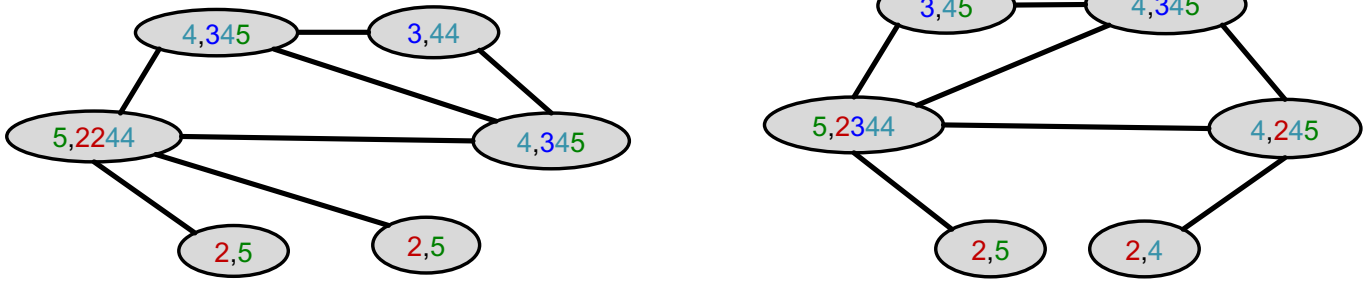




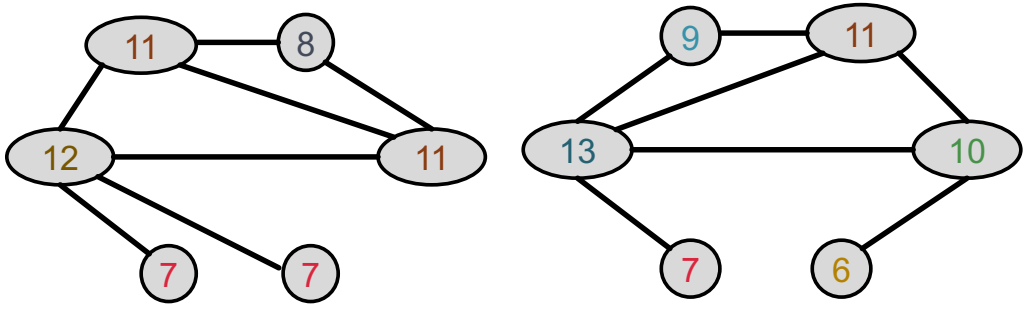
Weisfeiler-Lehman (WL) Kernel

Colour refinement: an example, given two graphs

- Aggregated colours



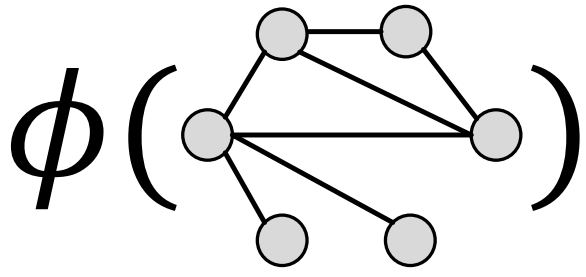
- Hash aggregated colours ($k = 2$)



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

Weisfeiler-Lehman (WL) Kernel

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

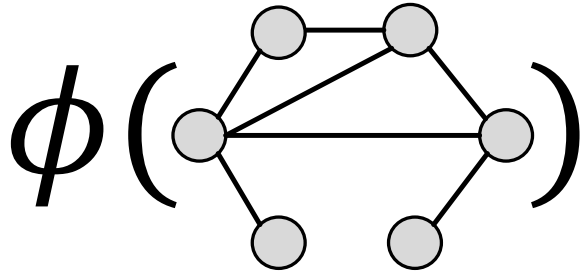


Colors

1,2,3,4,5,6,7,8,9,10,11,12,13

= [6,2,1,2,1,0,2,1,0, 0, 0, 2, 1]

Counts



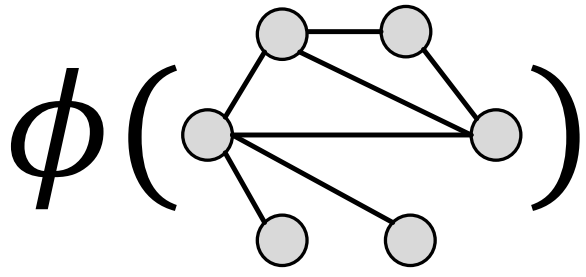
1,2,3,4,5,6,7,8,9,10,11,12,13

= [6,2,1,2,1,1,1,0,1, 1, 1, 0, 1]

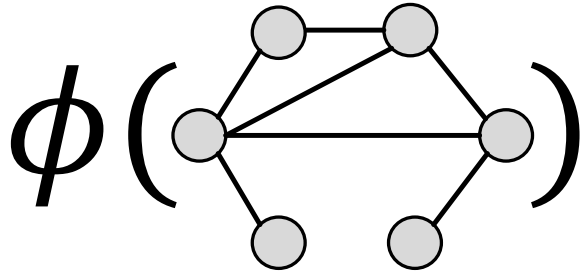
The WL Kernel value $K(G, G')$ is computed by the inner product of the colour count vectors.

Weisfeiler-Lehman (WL) Kernel

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



Colors
1,2,3,4,5,6,7,8,9,10,11,12,13
= [6,2,1,2,1,0,2,1,0, 0, 0, 2, 1]
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 $K(G, G')$ is computed by the inner product of the colour count vectors.

Graph-Level Features

- Graphlet Kernel
  - Graph is represented as bag-of-graphlets
  - Computationally expensive
- Weisfeiler-Lehman (WL) Kernel
  - Apply K-step colour refinement algorithm to enrich node colours
  - Different colours capture different K-hop neighbourhood structures
  - Graph is represented as bag-of-colours
  - Computationally efficient

Graph-Level Features

- Graphlet Kernel
  - Graph is represented as bag-of-graphlets
  - Computationally expensive
- Weisfeiler-Lehman (WL) Kernel
  - Apply K-step colour refinement algorithm to enrich node colours
  - Different colours capture different K-hop neighbourhood structures
  - Graph is represented as bag-of-colours
  - Computationally efficient

Machine Learning on Graphs

- Traditional ML pipeline (hand-crafted features + ML model)
- Hand-crafted features for graph data
  - Node-level: node degree, centrality, clustering coefficient, graphlets
  - Link-level: distance-based feature, local & global neighbourhood overlap
  - Graph-level: graphlet kernel, WL kernel