CS6208 : Advanced Topics in Artificial Intelligence
Graph Machine Learning

Lecture 4: Engineered and Shallow Learning of Graph Features

Semester 2 2022/23

Xavier Bresson

https://twitter.com/xbresson

Department of Computer Science
National University of Singapore (NUS)

Outline

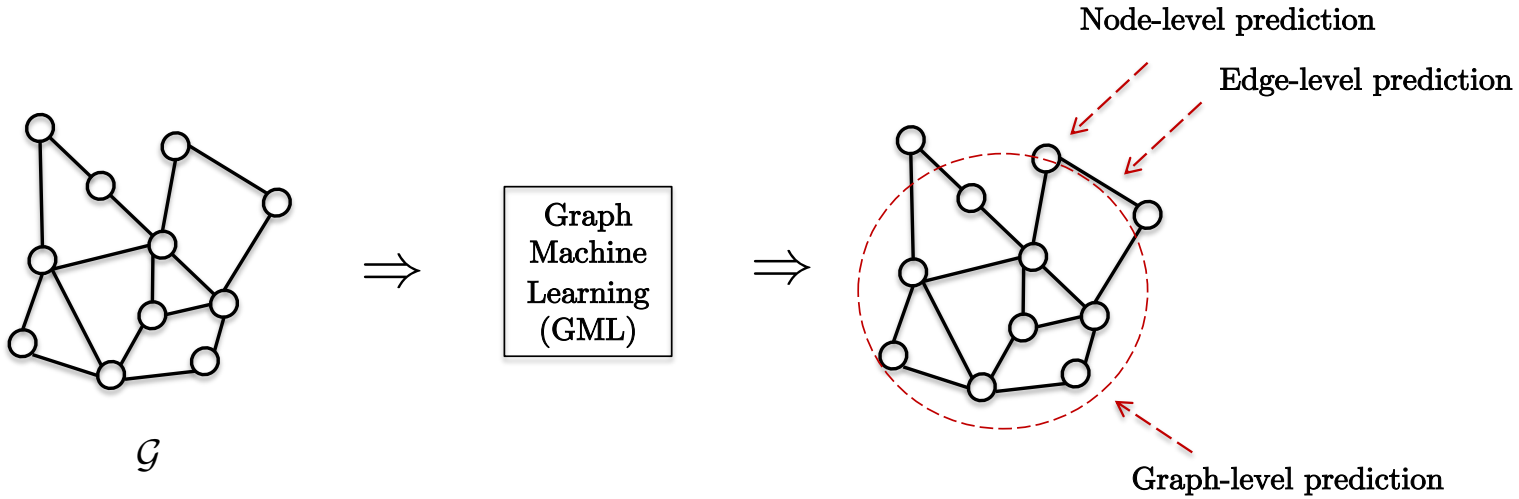
- Graph prediction tasks
- Graph feature without learning (engineered)
  - Node feature
  - Link feature
  - Graph feature
- Shallow graph feature learning
  - Node feature
  - Link feature
  - Graph feature
- Conclusion

Outline

- Graph prediction tasks
- Graph feature without learning (engineered)
  - Node feature
  - Link feature
  - Graph feature
- Shallow graph feature learning
  - Node feature
  - Link feature
  - Graph feature
- Conclusion

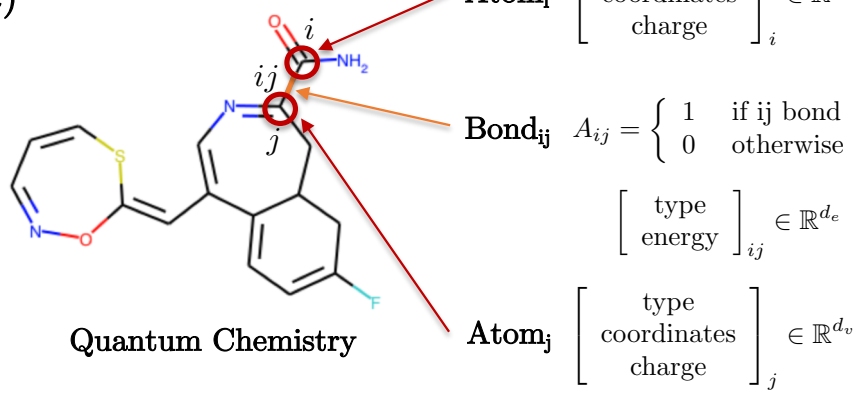
Graph prediction tasks

- There are three fundamental prediction tasks with graphs
  - Node-level prediction, e.g. fraudster in financial networks
  - Edge/link-level prediction, e.g. fraudulent transaction
  - Graph/subgraph-level prediction, e.g. laundering cluster



Graph features

- How to make node/edge/graph-level predictions ?
  - We need expressive, transferable/generalizable features.
- Type of (raw) features
  - Structural feature : node feature s.a. node degree, link feature e.g. shortest distance, graph feature s.a. graph kernel.
  - Raw data feature : node feature s.a. atom type, edge feature e.g. bond type, graph feature s.a. molecular energy.
- Development
  - Before 2014 : Engineered/hand-crafted features (e.g. pagerank)
  - 2014-2016 : Shallow learned features (e.g. node2vec)
  - After 2016 : Deep learned features (GNNs)

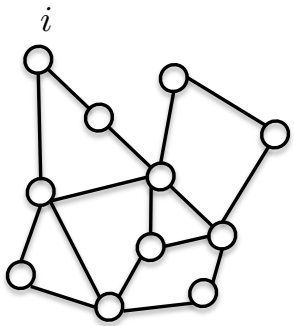


Outline

- Graph prediction tasks
- Graph feature without learning (engineered)
  - Node feature
  - Link feature
  - Graph feature
- Shallow graph feature learning
  - Node feature
  - Link feature
  - Graph feature
- Conclusion

Node-level feature

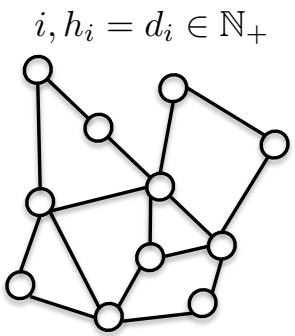
- There exist two properties to characterize the nodes in a graph.
  - Structural information, e.g. node degree
  - Positional/geometrical information, e.g. N-dim Euclidean embedding of nodes



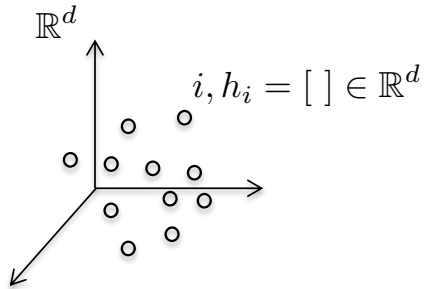
$\mathcal{G}$

Node feature

$\Rightarrow$



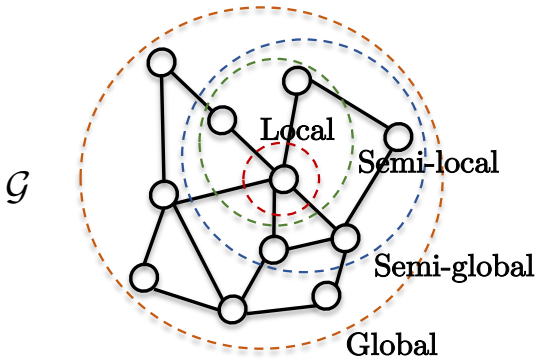
Structural information
e.g. node degree



Positional information
e.g. Euclidean embedding

Structural node feature

- Define the node feature from the graph topology/structure.
  - Local property of graph
    - Node degree : # neighbors connected to a node
  - Semi-local property of graph
    - Graphlet degree vector
  - Semi-global property of graph
    - Betweenness centrality
  - Global property of graph
    - Eigenvector centrality / PageRank

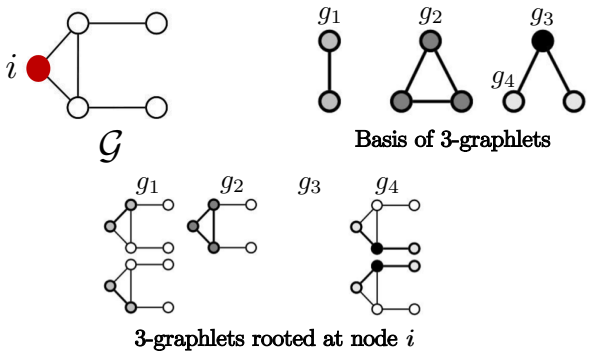


Graphlet degree vector[1]

- Graphlets form a basis of graph patterns, a.k.a. network motifs.
  - Any graph can be represented by a number of graphlets.
- The number of graphlets depends on the number of nodes composing the graphlet.
- Graphlet degree vector (GDV) : The vector of the number of graphlets rooted at the node.
  - This feature gives a detailed and complete description of the local neighbourhood topology.

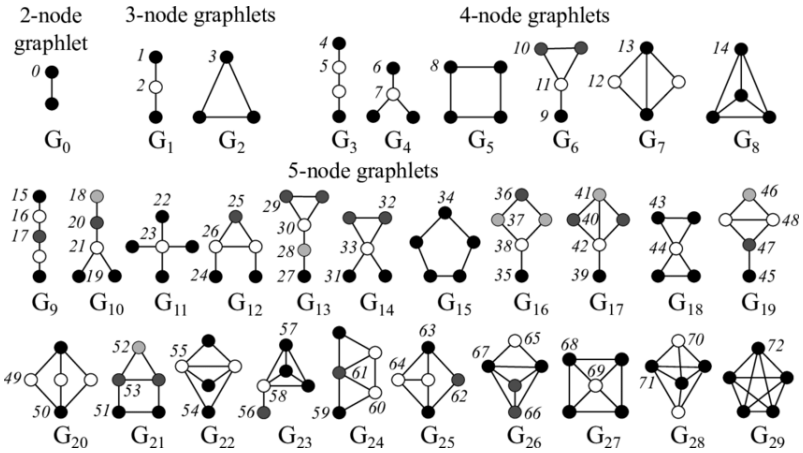
$$h_i = \#(g_k(i) \subset G) \text{ for } k = 1, ..., d = n_k$$

k^{th} graphlet rooted at node i



GDV of node i is
h_i=[2,1,0,2]
[#g1(i),#g2(i),#g3(i),#g4(i)]

Source: Jure Leskovec, CS224W



Source: Yaveroglu et al, Ergm.graphlets: A Package for ERG Modeling Based on Graphlet Statistics, 2014

[1] Przulj, Corneil, Jurisica, Modeling Interactome, Scale-Free or Geometric, 2004

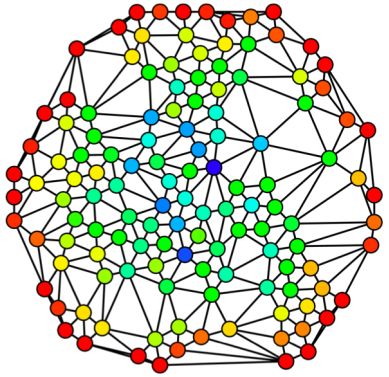
Betweenness centrality[1]

- Betweenness centrality : # shortest paths passing through the node
- Definition :

$$h_i = \sum_{s \neq v \neq t} \frac{\sigma_{\mathrm{st}}(i)}{\sigma_{\mathrm{st}}} \in \mathbb{R}_+$$

where σ_{st} is the # of shortest paths from node s to node t and $\sigma_{st}(i)$ is the # of those paths that pass through i.

- Interpretation : A node with high value is considered more important because more information could pass through that node.



Source: Wikipedia
Betweenness centrality value
(red is low and blue is large)

[1] Freeman, A set of measures of centrality based on betweenness, 1977

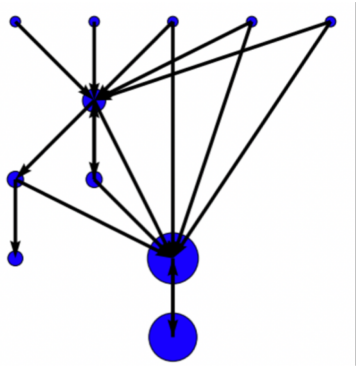
Eigenvector centrality

- Global property of graph
  - A measure of the influence of a node in a network.
  - A high score means that a node is connected to several nodes who themselves have high scores.
  - Several variants such as PageRank[1] with additional stochastic and irreducible matrix properties.
  - Task formulated as a spectral problem.
    - Perron-Frobenius theorem[2,3] : Eigenvector centrality is defined as the stationary state of the graph, characterized as the largest eigenvector (with eigenvalue 1) of the adjacency matrix.

$$Ah = h \in \mathbb{R}^N, \; h_i \in \mathbb{R}_+$$

Adjacency matrix of the graph

Largest eigenvector/ eigenvector centrality

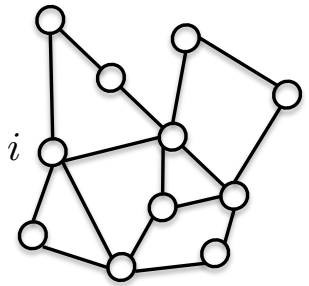


[1] Page, Brin, Motwani, Winograd, The PageRank citation ranking: Bringing order to the web, 1999
[2] Perron, Zur Theorie der Matrices, 1907
[3] Frobenius, Ueber Matrizen aus nicht negativen Elementen, 1912

Positional node feature

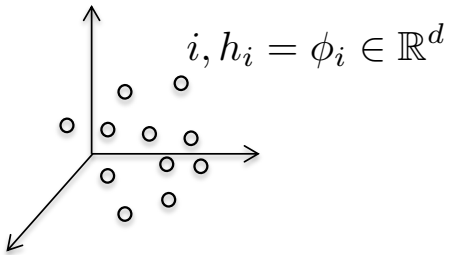
- Positional feature
  - Geometric features : Input/raw features s.a. 3-D coordinates or angles of atoms.
  - Topological features : Positional features computed from the graph structure.
    - Manifold learning : e.g. Laplacian eigenvectors[1] that embeds the graph into an d-dim Euclidean space while preserving the node proximity in the graph topology.

$$L = I - D^{-1/2} A D^{-1/2} = \Phi^T \Lambda \Phi$$

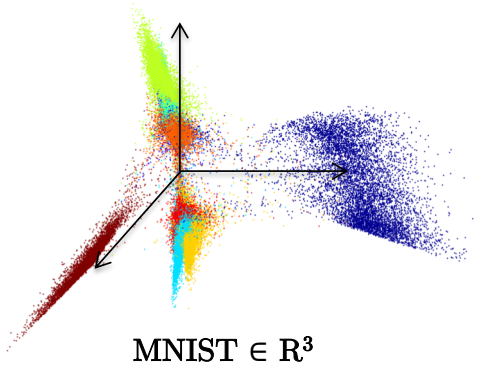


$\mathcal{G}$

$\Rightarrow$



Euclidean embedding with Laplacian eigenvectors



MNIST $\in \mathbb{R}^3$

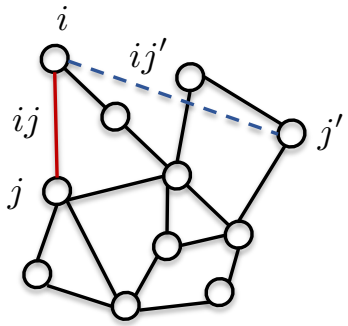
[1] Belkin, Niyogi, Laplacian eigenmaps for dimensionality reduction and data representation, 2003

Outline

- Graph prediction tasks
- Graph feature without learning (engineered)
  - Node feature
  - Link feature
  - Graph feature
- Shallow graph feature learning
  - Node feature
  - Link feature
  - Graph feature
- Conclusion

Link-level feature

- Edge/link is defined by a pair of nodes.

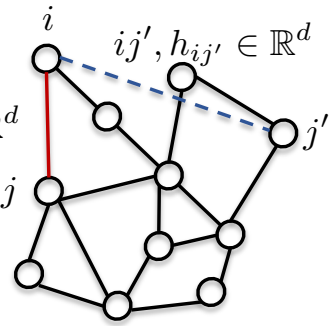


$\mathcal{G}$

Edge/link feature

$\Rightarrow$

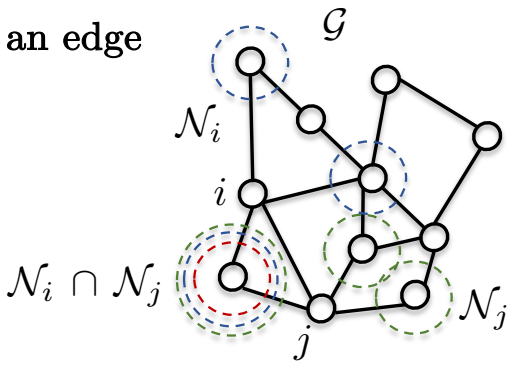
$ij, h_{ij} \in \mathbb{R}^d$



Edge-level feature

- Define the edge feature from the graph topology.
  - Local property of graph
    - Jaccard index[1] : # neighbor nodes shared between the two nodes of an edge

$$h_{ij} = \frac{\mathcal{N}_i \cap \mathcal{N}_j}{\mathcal{N}_i \cup \mathcal{N}_j} \in \mathbb{R}_+$$



$$h_{ij} = \frac{1}{5}$$

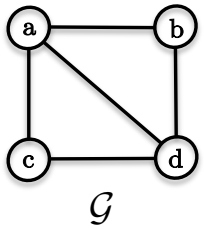
  - Semi-global property of graph
    - Katz index : # walks between the two nodes
    - Shortest path between the two nodes
    - Diffusion distance s.a. random walk between the two nodes

[1] Jaccard, The Distribution of the Flora in the Alpine Zone, 1912

Katz index[1]

- The number of walks between the two nodes can be computed with the binary adjacency matrix, i.e. $A_{ij} \in \{0,1\}$.
  - A^k : # walks of length k between two nodes

$$h_{ij} = A^k_{ij} \in \mathbb{N}_+$$



$\mathcal{G}$

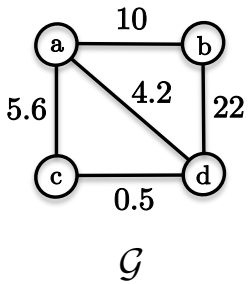
$$A = A^1 = \begin{array}{c} \begin{matrix} \text{a} & \text{b} & \text{c} & \text{d} \end{matrix} \\ \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix} \end{array} \begin{matrix} \text{(node index)} \\ \text{a} \\ \text{b} \\ \text{c} \\ \text{d} \end{matrix}$$

$$A^2 = A^1.A^1 = \begin{array}{c} \begin{matrix} \text{a} & \text{b} & \text{c} & \text{d} \end{matrix} \\ \begin{bmatrix} 3 & 1 & 1 & 2 \\ 1 & 2 & 2 & 1 \\ 1 & 2 & 2 & 1 \\ 2 & 1 & 1 & 3 \end{bmatrix} \end{array} \begin{matrix} \\ \text{a} \\ \text{b} \\ \text{c} \\ \text{d} \end{matrix}$$

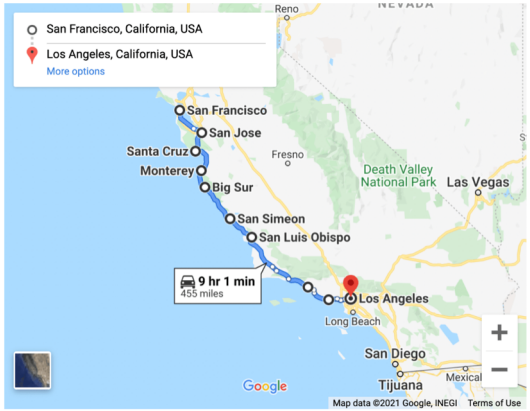
[1] Katz, A New Status Index Derived from Sociometric Analysis, 1953

Shortest path

- Edge feature is the shortest path between two nodes in a network.
  - Shortest distance computed with any adjacency matrix (binary and real-valued).
  - Dijkstra's algorithm[1] is linear O(E) with Fibonacci heap min-priority queue.



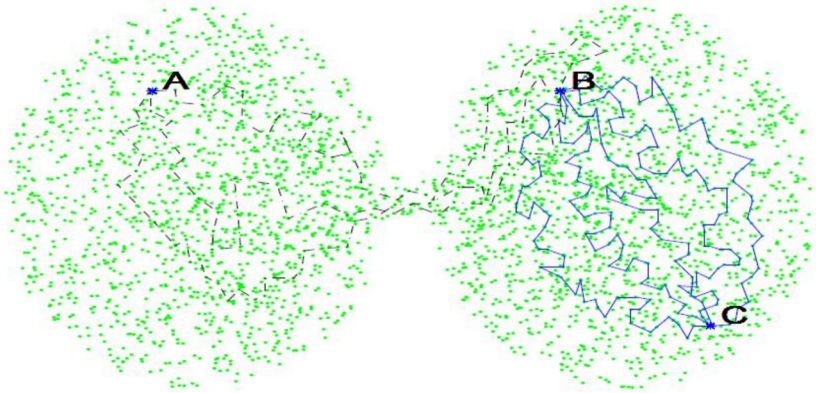
$\mathcal{G}$



[1] Dijkstra, A note on two problems in connexion with graphs, 1959

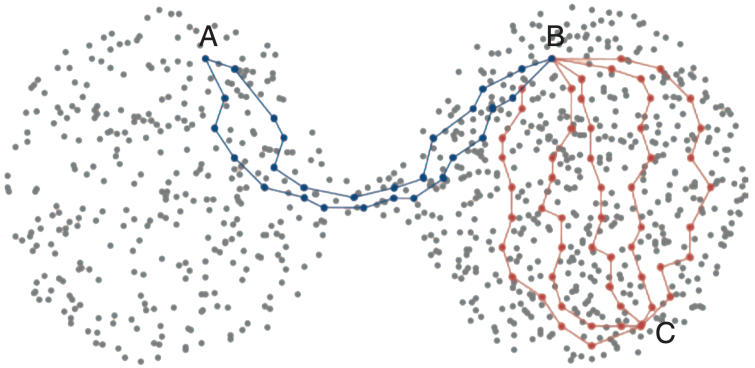
Diffusion distance

- Edge feature is given by the diffusion distance between two nodes in a network.
  - Random walk distance[1]
  - Heat/Laplacian diffusion distance[2]



$$d_{AB}^{\text{Diffusion}} \gg d_{BC}^{\text{Diffusion}}$$

Diffusion distance
Source[3]



$$d_{AB}^{\text{Shortest}} \approx d_{BC}^{\text{Shortest}}$$

Shortest path
Source[4]

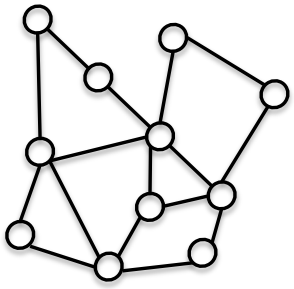
[1] Page, Brin, Motwani, Winograd, The PageRank citation ranking: Bringing order to the web, 1999
[2] Kondor and Vert, Diffusion kernels, 2004
[3] Rotbart, Diffusion maps, 2012
[4] Talmon, Cohen, Gannot, Coifman, Diffusion Maps for Signal Processing, 2013

Outline

- Graph prediction tasks
- Graph feature without learning (engineered)
  - Node feature
  - Link feature
  - Graph feature
- Shallow graph feature learning
  - Node feature
  - Link feature
  - Graph feature
- Conclusion

Graph-level feature

- Graph signature : Define the whole graph structure



$\mathcal{G}$

Graph feature

$\Rightarrow$

$h_G \in \mathbb{R}^d$

- Graph kernel : We use linear kernel techniques to define graph-level feature and distance between between two graphs
  - Graphlet count vector and kernel[1]
  - Weisfeiler-Lehman color vector and kernel[2]

[1] Shervashidze, Vishwanathan, Petri, Mehlhorn, Borgwardt, Efficient graphlet kernels for large graph comparison, 2009
[2] Shervashidze, Schweitzer, Van Leeuwen, Mehlhorn, Borgwardt, Weisfeiler-lehman graph kernels, 2011

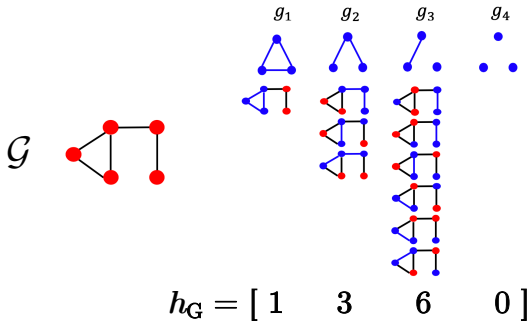
Graphlet count vector[1]

- Given a set of size-k graphlets, count the number of graphlets in a graph, a.k.a. bag of graphlets.

$$h_{Gi} = \#(g_i \subset G) \text{ for } i = 1, ..., d = n_k$$

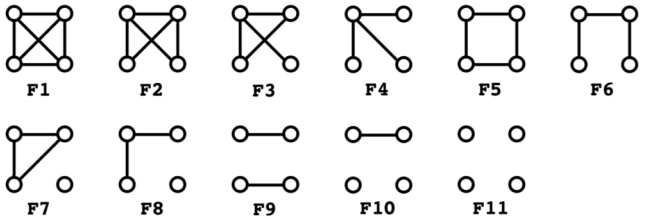
cardinality

size-k graphlets



All size-3 graphlets

Source: Jure Leskovec, CS224W



All size-4 graphlets[1]
Note that graphlets do not need to be connected (i.e. isolated nodes)

[1] Shervashidze, Vishwanathan, Petri, Mehlhorn, Borgwardt, Efficient graphlet kernels for large graph comparison, 2009

Graphlet kernel[1]

- Given two graphs, the kernel distance is defined as

$$K(\mathcal{G}, \mathcal{G}') = \langle h_{\mathcal{G}}, h_{\mathcal{G}'} \rangle \in \mathbb{N}_+$$

- Computational bottleneck
  - Counting size-k graphlets for a graph of size N is exponential $O(N^k)$, so intractable in practice for large k values.

[1] Shervashidze, Vishwanathan, Petri, Mehlhorn, Borgwardt, Efficient graphlet kernels for large graph comparison, 2009

Weisfeiler-Lehman color vector[1,2]

- Represent a graph as a bag of colors computed by an iterative coloring process until no new color can be created.

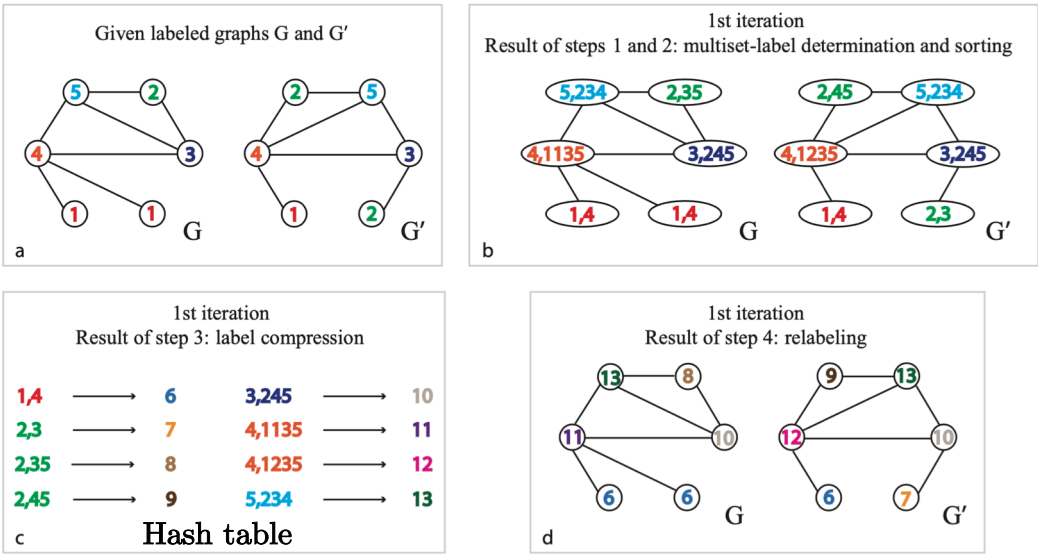
$$
\begin{aligned}
c_i^{k+1} &= f_{\text{coloring}}(c_i^k, \{c_j^{k+1}\}_{j \in \mathcal{N}_i}) \in \mathbb{R} \\
&= f_{\text{Hash}}(c_i^k, \{c_j^{k+1}\}_{j \in \mathcal{N}_i}) \in \mathbb{R} \\
h_{Gc} &= \#(i = c) \text{ for } c = 1, ..., d = n_{\text{color}}
\end{aligned}
$$

$$h_G = (2, 1, 1, 1, 1, 2, 0, 1, 0, 1, 1, 0, 1)$$

$$h_{G'} = (1, 2, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1)$$

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

Dictionary of created colors



Source[2]

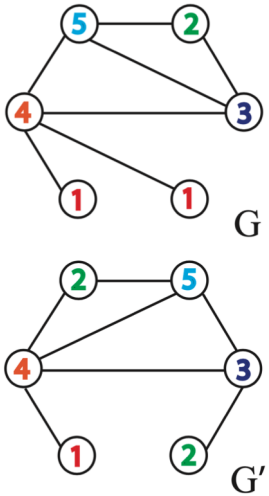
[1] Weisfeiler, Lehman, A reduction of a graph to a canonical form and an algebra arising during this reduction, 1968
[2] Shervashidze, Schweitzer, Van Leeuwen, Mehlhorn, Borgwardt, Weisfeiler-lehman graph kernels, 2011

Weisfeiler-Lehman kernel[1]

- Given two graphs, the kernel distance is defined as

$$K(\mathcal{G}, \mathcal{G}') = \langle h_{\mathcal{G}}, h_{\mathcal{G}'} \rangle \in \mathbb{N}_+$$

- Complexity
  - Linear O(E) as the coloring process uses all nodes and their neighbors.



$$h_G = (2, 1, 1, 1, 1, 2, 0, 1, 0, 1, 1, 0, 1)$$

$$h_{G'} = (1, 2, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1)$$

$$K(\mathcal{G}, \mathcal{G}') = \langle h_{\mathcal{G}}, h_{\mathcal{G}'} \rangle = 11$$

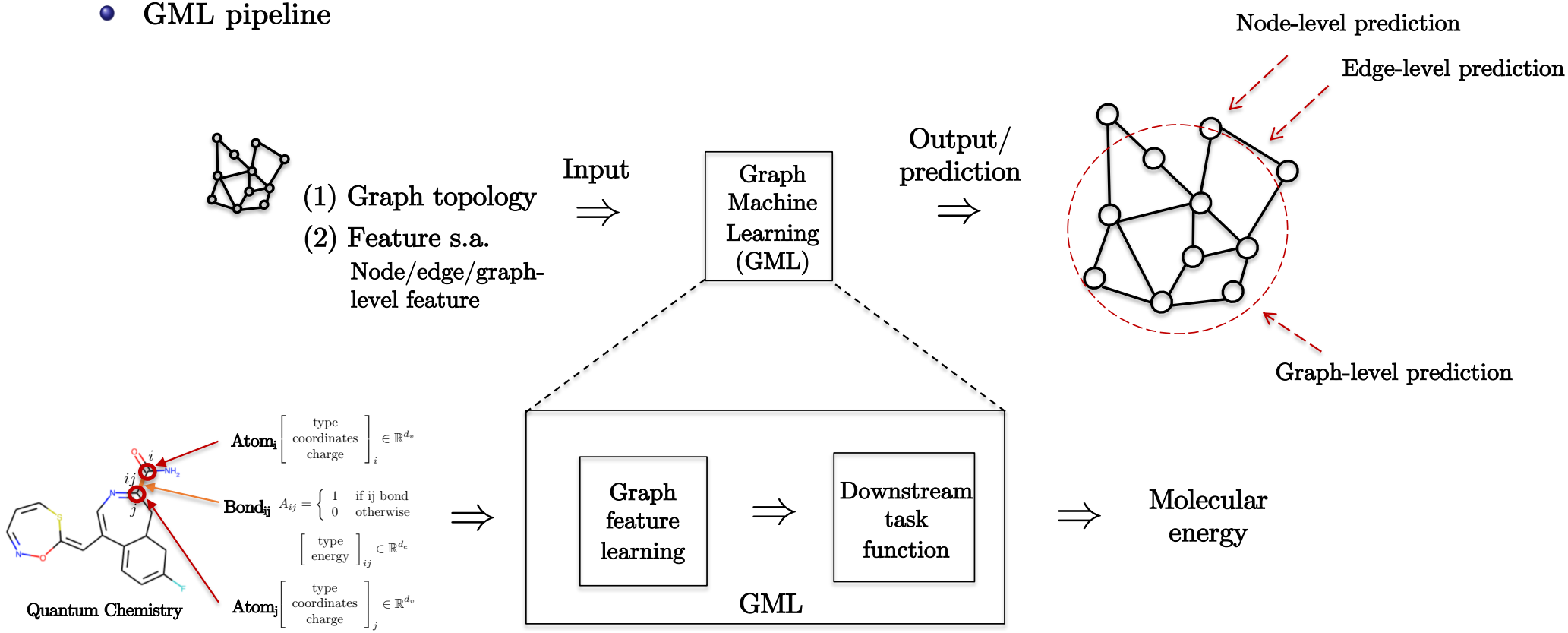
[1] Shervashidze, Schweitzer, Van Leeuwen, Mehlhorn, Borgwardt, Weisfeiler-lehman graph kernels, 2011

Outline

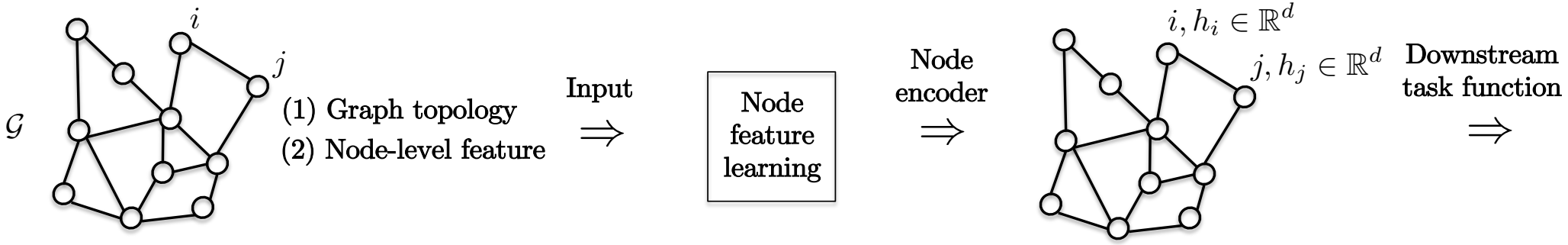
- Graph prediction tasks
- Graph feature without learning (engineered)
  - Node feature
  - Link feature
  - Graph feature
- **Shallow graph feature learning**
  - **Node feature**
  - Link feature
  - Graph feature
- Conclusion

Graph feature learning

- GML pipeline

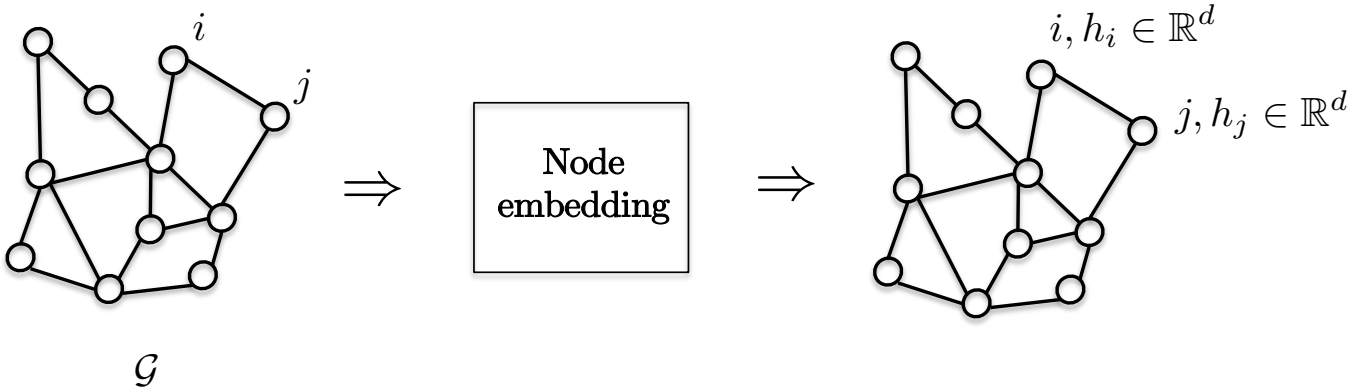


Node feature learning



Shallow node feature learning

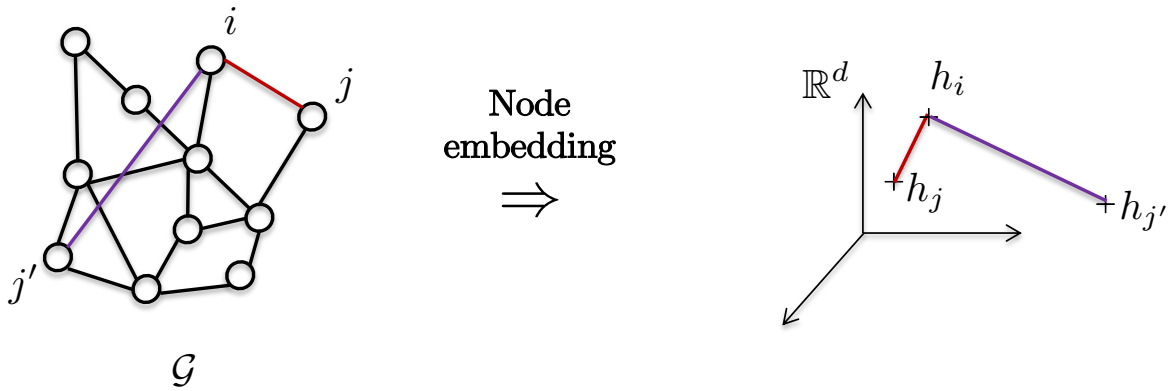
- Several (restrictive) hypothesis
  - Shallow learning : Single hidden layer, a.k.a. node embedding
  - Lack of raw node feature (only adjacency matrix/graph topology)



- How to compute $h_i \in R^d$?

Fundamental embedding property

- The embedding space must preserve the node proximity on G.
  - A pair of nodes (i,j) close on G must produce a pair of vectors (h_i,h_j) close in R^d.



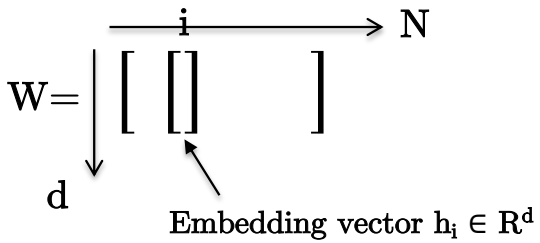
Embedding similarity

- Closeness/similarity between two vectors can be defined with
  - Euclidean distance : $s_{ij} = ||h_i - h_j||_2^2 \in R_+$
  - Scalar product : $s_{ij} = \langle h_i, h_j \rangle$ (or $h_i^T h_j$) $\in R$
  - Computational complexity is $O(d)$.
  - Note that Euclidean distance = scalar product for L^2-normalized vectors i.e. $||h_i||_2 = 1$
- More distances can be used s.a. optimal transport/Wasserstein distance[1] but the computational complexity is usually higher $O(d^2)$.

[1] Cuturi, Sinkhorn distances: Lightspeed computation of optimal transport, 2013

Shallow encoding

- Simplest feature learning is as follows
  - Map node index $i \in \{0,1,...,N-1\}$ to a one-hot encoding vector $1_i = [0,0,..0,1,0,...,0] \in R^N$.
  - Linear transform vector 1_i to $h_i = W1_i \in R^d$, with $W \in R^{d \times N}$.
- Interpretation : Matrix W is a learnable lookup table !

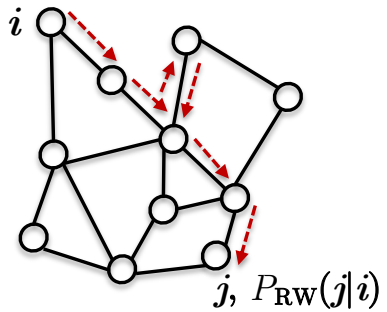


Shallow encoding

- How do we compute/learn parameters W ?
- Standard learning technique
  - Select a similarity distance between two node embeddings.
  - Design a loss function that enforces the embedding similarity.
  - Find the W parameters by gradient descent optimization.

Node-to-vectors

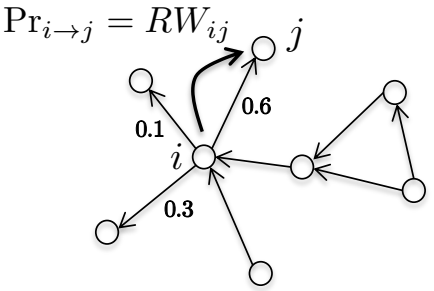
- Motivation is to extend word2vec[1] from NLP to graphs.
  - DeepWalk[2]
  - Node2vec[3]
- Core idea : Use random walks (RWs) to explore the topology of a graph.
- This exploration is stochastic, which is an advantage and a limitation.
  - An advantage because sampling a RW is fast and can be done in parallel.
  - A limitation because the number of RWs to explore the whole topology can be intractable (they are also biased to explore local topology).
- RWs find nodes closely connected on graphs : The proximity between two nodes i and j on a graph is given by the probability $\mathrm{P_{RW}}(j|i)$ of visiting node j starting from node i with a RW path.



RW starting at i and visiting j after 6 steps/hops.

[1] Mikolov, Sutskever, Chen, Corrado, Dean, Distributed representations of words and phrases and their compositionality, 2013
[2] Perozzi, Al-Rfou, Skiena, Deepwalk: Online learning of social representations, 2014
[3] Grover, Leskovec, node2vec: Scalable feature learning for networks, 2016

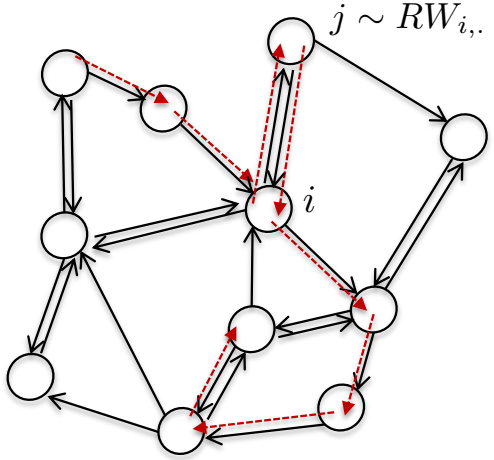
Random walk



- How to generate a RW path?
  - We first define the random walk probability transition matrix as

$$RW = D^{-1}A, \quad D_{ii} = \sum_j A_{ij}$$

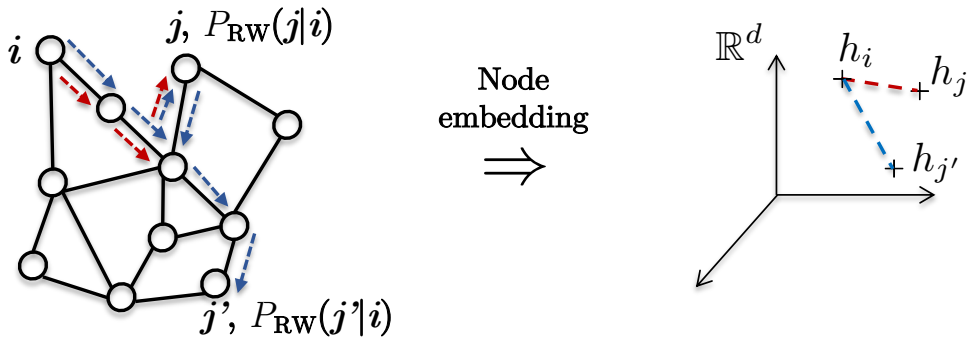
  - Each row of RW, i.e. $\mathrm{RW}_{\mathrm{i},.}$, corresponds to the probability to move from vertex i to vertex j.
  - A RW path is obtained by iteratively sampling the RW matrix with a Bernouilli distribution, i.e. given a node i, sample the next node $j \sim \mathrm{RW}_{\mathrm{i},.}$, assign j to i, iterate.



Sampling a RW path (⇢) in a graph

Modeling RW probability

- How to design $P_{RW}(j|i)$ w.r.t. the node embeddings h_i , $h_j \in R^d$?
  - For (i,j) close on G, we have (h_i,h_j) close in R^d.
  - For (i,j) close on G, we have high probability $P_{RW}(j|i)$.
  - Consequence : High probability $P_{RW}(j|i)$ implies similar (h_i,h_j), and inversely.



Modeling RW probability

- Different designs for $P_{RW}(j|i)$ are possible :

$$P_{\mathrm{RW}}(j|i) = f_W(h_i, h_j)$$
$$= \frac{\sigma(d_{ij})}{Z}$$

$$\sigma(\cdot) = \begin{cases} \exp(\cdot) \\ \mathrm{sigmoid}(\cdot) \end{cases}$$
Base function
(sparse softmax, dense softmax)

$$d_{ij} = \begin{cases} h_i^T h_j \\ \|h_i - h_j\|_2^2 \end{cases}$$
Similarity score
(dot product, Euclidean distance)

$$Z = \sum_{j' \in V} \sigma(d_{ij'})$$
Normalization constant
(partition function)

$$h_\eta = W 1_\eta, \eta = i, j$$
Node embedding with
learnable lookup table W
computed by optimization

Loss function

- We aim at maximizing the product of probabilities $P_{RW}(j|i)$ for all pairs of nodes (i,j) connected by all RWs, or equivalently minimizing the sum of - log $P_{RW}(j|i)$ (using the log function trick for fast and stable computation) :

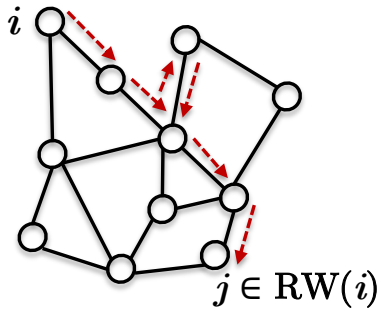
$$\max_W \mathbb{E}_{\mathrm{RW}}\Big(\log \Pi_{i\in V}\Pi_{j\in \mathrm{RW}(i)} P_{\mathrm{RW}}(j|i)\Big)$$

$$\min_W \; L(W) = \mathbb{E}_{\mathrm{RW}}\Big(-\sum_{i\in V}\sum_{j\in \mathrm{RW}(i)} \log P_{\mathrm{RW}}(j|i)\Big)$$

- Minimizing L forces all node embeddings h_j on the RW(i) to be close to the node embedding h_i.
- However, this loss estimation is computationally intractable !
  - We approximate using a single sample of RW[1] :

$$L(W) = -\sum_{i\in V}\sum_{j\in \mathrm{RW}(i)} \log P_{\mathrm{RW}}(j|i)$$

  - In practice, a fixed-length RW is used for efficiency and parallel implementation.



[1] Perozzi, Al-Rfou, Skiena, Deepwalk: Online learning of social representations, 2014

Algorithm

- Two-step algorithm
  - For each i in the graph, sample (in parallel) a path RW(i).
  - Minimize the likelihood probability loss :

$$
\begin{aligned}
L(W) &= -\sum_{i \in V} \sum_{j \in \mathrm{RW}(i)} \log P_{\mathrm{RW}}(j|i) \\
&= -\sum_{i \in V} \sum_{j \in \mathrm{RW}(i)} \log \frac{\sigma(d_{ij})}{Z} \\
&= -\sum_{i \in V} \sum_{j \in \mathrm{RW}(i)} \log \frac{\sigma(d_{ij})}{\sum_{j' \in V} \sigma(d_{ij'})} \\
&= -\sum_{i \in V} \sum_{j \in \mathrm{RW}(i)} \log \frac{\exp(h_i^T h_j)}{\sum_{j' \in V} \exp(h_i^T h_{j'})}, h_\eta = W 1_\eta, \eta = i, j, j'
\end{aligned}
$$

with $\sigma = \exp$ and $d_{ij} = h_i^T h_j$

Still too expensive to optimize as complexity is $O(N^2)$!

Negative sampling[1]

- Fast and popular technique to remove the partition function (denominator) in the probability.
  - Step 1 : Approximate the original problem with nodes outside the RW path :

$$\max_W \; L(W) = \log \Pi_{i \in V} \Pi_{j \in \mathrm{RW}(i)} P_{\mathrm{RW}}(j|i)$$

$$\Downarrow \quad \text{approximate}$$

$$\max_W \; L_{\mathrm{NS}}(W) = \log \Pi_{i \in V} \Pi_{j \in \mathrm{RW}(i)} \underbrace{\Pi_{k \notin \mathrm{RW}(i)}} \frac{P_{\mathrm{RW}}(j|i)}{P_{\mathrm{RW}}(k|i)}$$

Positive samples

Negative samples

Approximate with uniform random distribution over the nodes, $k \sim U_V$

  - Interpretation : The original probability of positive sample is re-inforced using the probability of negative sample (max ratio $P_{pos}/P_{neg} \Rightarrow P_{pos} > P_{neg}$).

[1] Mikolov, Sutskever, Chen, Corrado, Dean, Distributed representations of words and phrases and their compositionality, 2013

Negative sampling[1]

- **Step 2 : Change the log product into sum of log :**

$$\max_W \; L_{\mathrm{NS}}(W) = \log \Pi_{i \in V} \Pi_{j \in \mathrm{RW}(i)} \Pi_{k \sim U_V} \frac{P_{\mathrm{RW}}(j|i)}{P_{\mathrm{RW}}(k|i)}$$

$$= \sum_{i \in V} \sum_{j \in \mathrm{RW}(i)} \sum_{k \sim U_V} \log \frac{P_{\mathrm{RW}}(j|i)}{P_{\mathrm{RW}}(k|i)}$$

$$\text{with } P_{\mathrm{RW}}(j|i) = \frac{\sigma(d_{ij})}{Z}, P_{\mathrm{RW}}(k|i) = \frac{\sigma(d_{ik})}{Z}$$

$$= \sum_{i \in V} \sum_{j \in \mathrm{RW}(i)} \sum_{k \sim U_V} \log \sigma(d_{ij}) - \cancel{\log Z} - \log \sigma(d_{ij}) + \cancel{\log Z}$$

No more Z !

$$= \sum_{i \in V} \sum_{j \in \mathrm{RW}(i)} \sum_{k \sim U_V} \log \sigma(d_{ij}) - \log \sigma(d_{ik})$$

$$= \sum_{i \in V} \sum_{j \in \mathrm{RW}(i)} \sum_{k \sim U_V} \log \sigma(h_i^T h_j) - \log \sigma(h_i^T h_k), h_\eta = W 1_\eta, \eta = i, j, k$$

with $d_{ij} = h_i^T h_j$

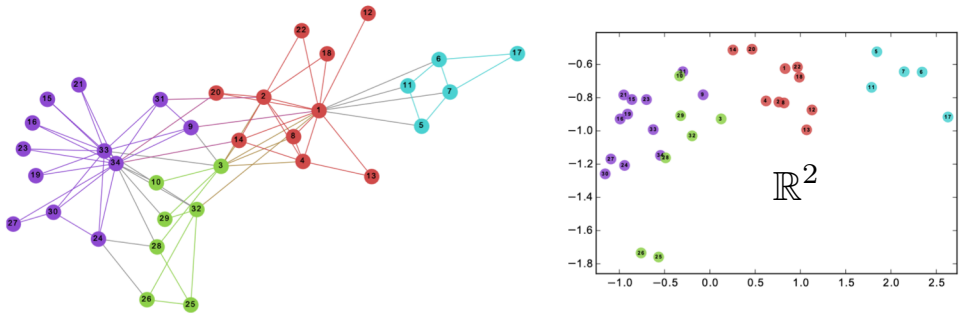
[1] Mikolov, Sutskever, Chen, Corrado, Dean, Distributed representations of words and phrases and their compositionality, 2013

Algorithm[1]

- Two-step algorithm
  - For each i in the graph, sample (in parallel) a path $\mathrm{RW}(i)$.
  - Minimize the likelihood probability loss with gradient descent :

$$\min_W \; L_{\mathrm{NS}}(W) = -\sum_{i \in V} \sum_{j \in \mathrm{RW}(i)} \sum_{k \sim U_V} \log \sigma(h_i^T h_j) - \log \sigma(h_i^T h_k), h_\eta = W 1_\eta, \eta = i, j, k$$

  - Complexity is linear O(N)
  - Result on Zachary's karate club dataset



(a) Input: Karate Graph (b) Output: Representation

Source[1]

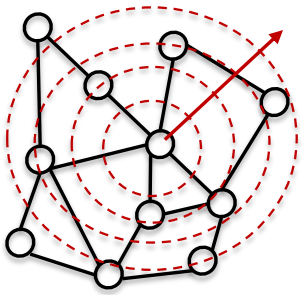
[1] Perozzi, Al-Rfou, Skiena, Deepwalk: Online learning of social representations, 2014

Node2vec[1]

- RW samples the graph geometry without guidance.
  - It is mathematically valid but this process requires a large number of RW paths to approximate well the original loss :

$$L(W) = \mathbb{E}_{\mathrm{RW}}\Big(-\sum_{i \in V} \sum_{j \in \mathrm{RW}(i)} \log P_{\mathrm{RW}}(j|i) \Big)$$

- The stochastic exploration of the graph topology can be improved with better RW paths on the network, first by exploring locally and then expanding globally.



[1] Grover, Leskovec, node2vec: Scalable feature learning for networks, 2016

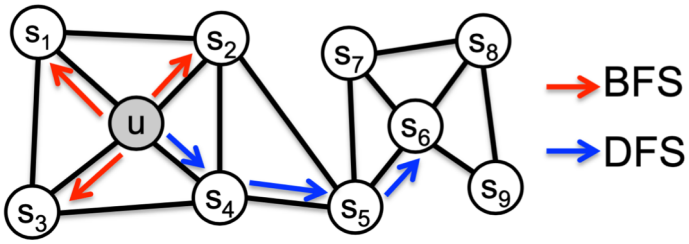
Biased RW path

- We can control a new RW with three explorative actions.
  - Local exploration with BFS (breadth-first search)
  - Global exploration with DFS (depth-first search)
  - No exploration (move back to the previous node)
- We sample the next action.
  - Use a parameter q to trade-off BFS and DFS.
  - Use another parameter p to move back to previous node.
  - Probability of actions sampled by Bernoulli :

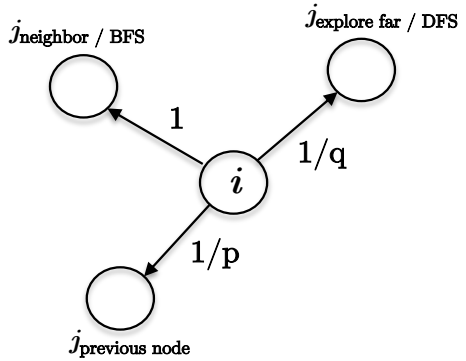
$$P_{\text{action}}(j|i) = \frac{1}{c} \begin{bmatrix} 1/p \\ 1 \\ 1/q \end{bmatrix} \begin{matrix} \text{Back to previous node} \\ \text{DFS} \\ \text{BFS} \end{matrix}$$

Normalization constant

- Keep linear complexity O(N)



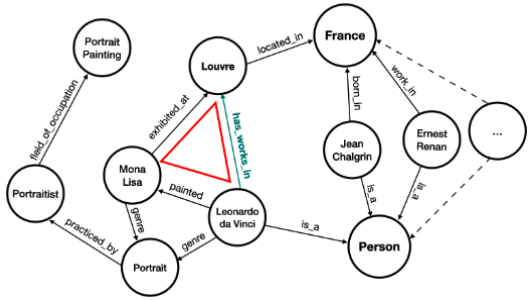
Source[1]



[1] Grover, Leskovec, node2vec: Scalable feature learning for networks, 2016

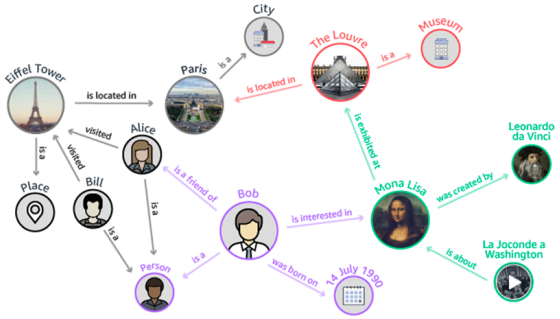
Summary

- Learning node-level feature
  - Use graph topology : Two node embeddings similar/close if the two nodes on the graph are topologically close w.r.t. RW path.
  - The choice of closeness can be adapted to different tasks, with RW replaced with e.g. shortest path (molecular science), or other diffusion process like spectral distance.
  - The learning is shallow, a single hidden layer, which limits the feature expressivity.
  - No raw feature has been used, only the graph topology has been considered so far.
  - The node embeddings are learned for a given graph and cannot be transferred to a new graph.



Source[1]

No transfer learning possible



Source[2]

[1] Hebatallah et-al, Locality-aware subgraphs for inductive link prediction in knowledge graphs, 2023
[2] https://aws.amazon.com/neptune/knowledge-graphs-on-aws

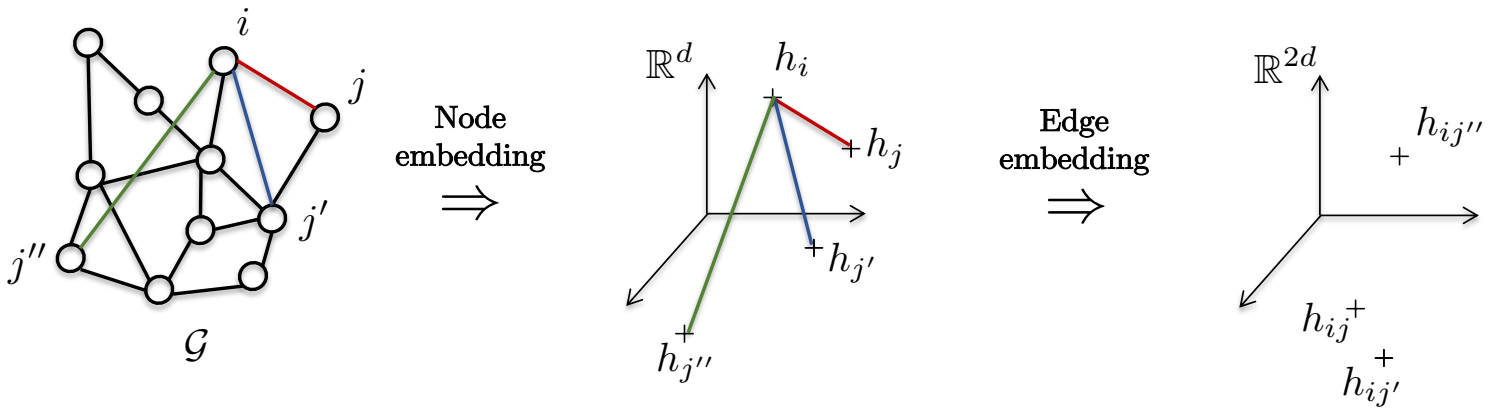
Outline

- Graph prediction tasks
- Graph feature without learning (engineered)
  - Node feature
  - Link feature
  - Graph feature
- Shallow graph feature learning
  - Node feature
  - Link feature
  - Graph feature
- Conclusion

Link-level features

- Link embedding
  - A simple feature that works well in practice is to concatenate a pair of pre-trained node embedding.

$$h_{ij} = \text{Concat}(h_i, h_i) \in \mathbb{R}^{2d}, \quad h_\eta = W_1 1_\eta, \eta = i, j$$



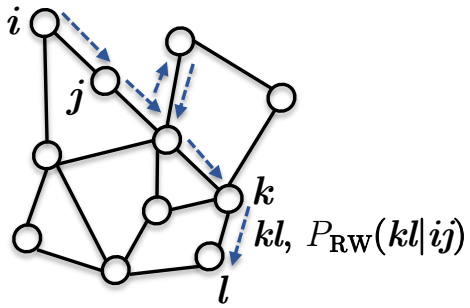
Link-level features

- Link embedding
  - RW embedding of graphs
  - Compute link embedding by predicting the link kl to be close to the link ij in the RW path starting from i.

$$L(W) = \mathbb{E}_{\text{RW}}\Big(-\sum_{i \in V} \sum_{kl \in \text{RW}(i)} \log P_{\text{RW}}(kl|ij) \Big)$$

$$h_{\eta\gamma} = W_2 \text{Concat}(W_1 1_\eta, W_1 1_\gamma) \in \mathbb{R}^d, W_1 \in \mathbb{R}^{d \times N}, W_2 \in \mathbb{R}^{d \times 2d}, \eta = i, k, \gamma = j, l$$

$$P_{\text{RW}}(ij|kl) = \frac{\exp(h_{ij}^T h_{kl})}{\sum_{k'l' \in V^2} \exp(h_{ij}^T h_{k'l'})}$$



Outline

- Graph prediction tasks
- Graph feature without learning (engineered)
  - Node feature
  - Link feature
  - Graph feature
- Shallow graph feature learning
  - Node feature
  - Link feature
  - Graph feature
- Conclusion

Graph-level feature

- Graph embedding
  - A simple feature that works well in practice is mean or sum of pre-trained node embeddings :

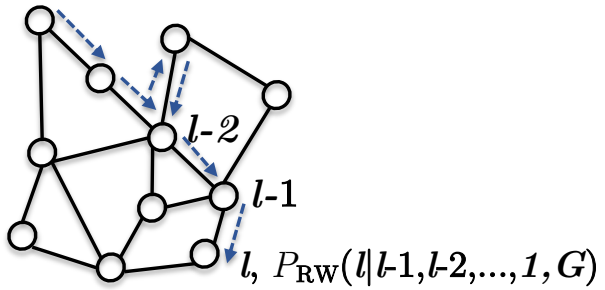
$$h_G = \frac{1}{N} \sum_{i \in V} h_i \in \mathbb{R}^d$$

  - RW embedding of graphs[1] : Compute graph embedding by predicting RW paths of length L, L+1,...,L+M given RW paths of smaller lengths.

$$L(W) = \mathbb{E}_{\text{RW}} \Big(- \sum_{l=L}^{L+M} \log P_{\text{RW}}(l | l-1, l-2, ..., 1, G) \Big)$$

$$\text{with } h_G = w \in \mathbb{R}^d$$

Learnable vector



[1] Ivanov, Burnaev, Anonymous walk embeddings, 2018

Outline

- Graph prediction tasks
- Graph feature without learning (engineered)
  - Node feature
  - Link feature
  - Graph feature
- Shallow graph feature learning
  - Node feature
  - Link feature
  - Graph feature
- **Conclusion**

Conclusion

- Engineered graph feature
  - Limited expressivity, domain specific (expert knowledge)
- Shallow learned feature
  - Extension of word2vec to graphs : Pre-trained node embedding
  - Limitations
    - One hidden layer (but deeper is better for expressivity)
    - No use of input raw node/edge/graph features (missing critical information)
    - Learned features are not transferable to a new graph (generalization failure)

Questions?