Applied Machine Learning

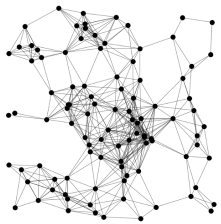
Machine Learning with Graphs

Reihaneh Rabbany

McGill
School of Computer Science

Learning objectives

- How to represent graph structured data
- Unsupervised learning with graphs
    - Community detection (clustering)
- Supervised learning with graphs
    - Node classification



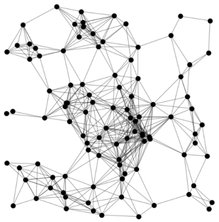
Motivation

Our world is **complex** and analyzing interconnected data provides the much needed tools to study today's phenomena (e.g., online societies) and enables us to address the world's emerging problems (e.g., covid-19)

Complex Systems

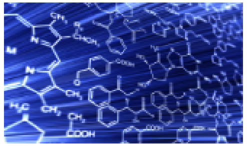
- consists of many interconnected parts
- characterized by time-dependent interactions among their parts
- not an aggregation of their separate parts
- when looked at as a whole gives non trivial insights
- often interactions change states of parts, and the states of the parts change the networks of interactions

Motivation: applications

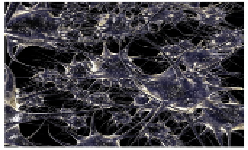


natural sciences: connections between atoms, molecules, cells, organisms and even the cosmic web

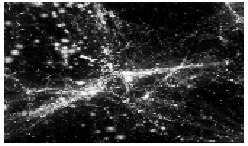
Chemistry



Biology

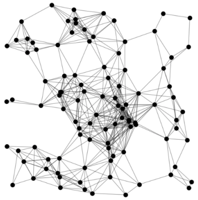


Physics



from a demo of galaxy networks

applied sciences: looking at compex system, as a whole, gives us non trivial insights and is necessary to understand these systems in many applications, e.e. in Medicine, law, even culinary (check this flavor network)



Representing **Interconnected Data**

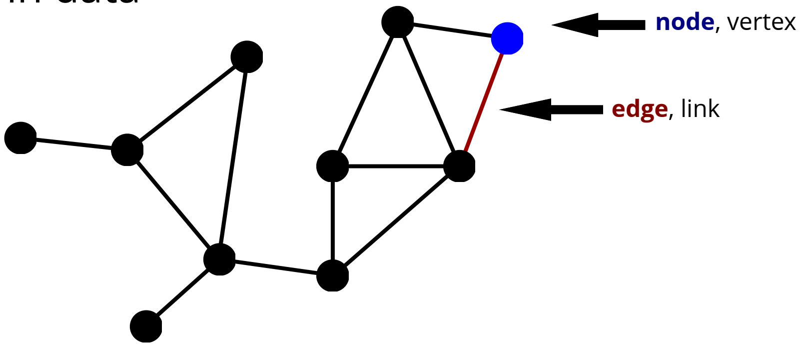
we used independent **instances** as data in this course:

$$X = \begin{bmatrix} {x^{(1)}}^\top \\ {x^{(2)}}^\top \\ \vdots \\ {x^{(N)}}^\top \end{bmatrix} = \begin{array}{cl} \begin{bmatrix} x_1^{(1)}, & x_2^{(1)}, & \cdots, & x_D^{(1)} \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{(N)}, & x_2^{(N)}, & \cdots, & x_D^{(N)} \end{bmatrix} & \begin{array}{l} \text{one instance} \\ \\ \\ \end{array} \\ \begin{array}{cccc} \quad & \text{one feature} & \quad & \quad \end{array} & \end{array} \in \mathbb{R}^{N \times D}$$

Graphs represent the relation between **instances** in data

the default representation

- variations: **simple**, weighted, directed, signed, multi-edges and multi-type nodes (heterogenous), attributed (nodes and or edges have feature vectors), dynamic (sequence of graphs), multilayer networks (multi-view), hypergraphs (beyond pairwise relations), etc.



Representing **Graphs**

Features Matrix
node features

$$X = \begin{bmatrix} x_1^{(1)}, & x_2^{(1)}, & \cdots, & x_D^{(1)} \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{(N)}, & x_2^{(N)}, & \cdots, & x_D^{(N)} \end{bmatrix} \in \mathbb{R}^{N \times D}$$

one instance

one feature

Adjacency Matrix
connections between nodes

marginals of A are called **degree**

$d_i = \sum_j A_{ij}$

$$A = \begin{bmatrix} A_{11}, & A_{12}, & \cdots, & A_{1N} \\ \vdots & \vdots & \ddots & \vdots \\ A_{N1}, & A_{N2}, & \cdots, & A_{NN} \end{bmatrix} \in \mathbb{R}^{N \times N}$$

inlinks

all nodes linking to 1

if unweighted then $\in \{0,1\}^{N \times N}$

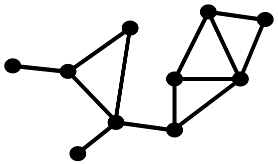
outlinks

all nodes node 2 links to

Real world graphs are sparse (have lots of zeros) and we use sparse matrix representations to store them (only store non-zero values)

Representing Graphs

Adjacency Matrix



- person & friendship
- paper & citation
- cities & train tracks
- protiens & binding

$$A = \begin{array}{cl} \begin{bmatrix} A_{11}, & A_{12}, & \cdots, & A_{1N} \\ \vdots & \vdots & \ddots & \vdots \\ A_{N1}, & A_{N2}, & \cdots, & A_{NN} \end{bmatrix} & \begin{array}{l} \text{inlinks} \\ \\ \\ \end{array} \\ \quad\quad\quad \text{outlinks} & \end{array}$$

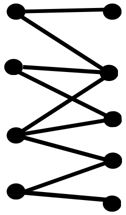
all nodes linking to 1

all nodes node 2 links to

$\in \mathbb{R}^{N \times N}$

if unweighted then $\in \{0,1\}^{N \times N}$

Incidence Matrix



often used to represent **bipartite** graphs

- actor & movies
- authors & papers
- metabolites & reactions
- words & documents
- two possible one mode projections: $B^\top B$, and $BB^\top$

$$B = \begin{array}{cl} \begin{bmatrix} A_{11}, & A_{12}, & \cdots, & A_{1M} \\ \vdots & \vdots & \ddots & \vdots \\ A_{N1}, & A_{N2}, & \cdots, & A_{NM} \end{bmatrix} & \begin{array}{l} \text{edges, or second set of nodes} \\ \\ \\ \end{array} \\ \quad\quad\quad \text{nodes} & \end{array}$$

all edges node 1 belongs to

all nodes edge 2 links

$\in \mathbb{R}^{N \times M}$

if unweighted then $\in \{0,1\}^{N \times M}$

Representing Graphs

Laplacian Matrix

$$\underbrace{L = D - A}_{\in \mathbb{R}^{N \times N}}$$

diagonal degree matrix

$$A = \begin{bmatrix} A_{11}, & A_{12}, & \cdots, & A_{1N} \\ \vdots & \vdots & \ddots & \vdots \\ A_{N1}, & A_{N2}, & \cdots, & A_{NN} \end{bmatrix} \in \mathbb{R}^{N \times N}$$

inlinks

all nodes linking to 1

if unweighted then $\in \{0,1\}^{N \times N}$

outlinks

all nodes node 2 links to

Eigenvalues of Graph laplacian tells us about the connectivity of the graph

- e.g. number of zero eigenvalues is the number of connected components
- second-smallest eigenvalue of L is called Algebraic connectivity or Fiedler value
- **Signs of values in Fiedler eigenvector** (associated to Fiedler eigenvalue) tell us how to partition the graph into two components by breaking least edges, i.e. **minimum cut** solution

$$L = \begin{bmatrix} d_1, & -A_{12}, & \cdots, & -A_{1N} \\ \vdots & \vdots & \ddots & \vdots \\ -A_{N1}, & -A_{N2}, & \cdots, & d_N \end{bmatrix} \in \mathbb{R}^{N \times N}$$

sums to zero

sums to zero

Representing Graphs

Adjacency Matrix

connections between nodes

marginals of A are called **degree**

$d_i = \sum_j A_{ij}$

$$A = \begin{array}{cc} \begin{bmatrix} A_{11}, & A_{12}, & \cdots, & A_{1N} \\ \vdots & \vdots & \ddots & \vdots \\ A_{N1}, & A_{N2}, & \cdots, & A_{NN} \end{bmatrix} & \begin{array}{l} \text{inlinks} \\ \\ \\ \end{array} \\ \begin{array}{cccc} & \text{outlinks} & & \end{array} & \end{array} \in \mathbb{R}^{N \times N}$$

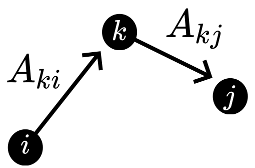
all nodes linking to 1

all nodes node 2 links to

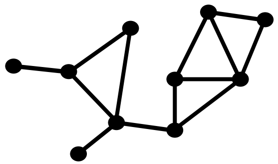
if unweighted then $\in \{0,1\}^{N \times N}$

Powers of A

- A^2 : # of walks with length two
  - If undirected, number of common neighbors
  - what is A^2_{ii}?
- A^3 : # of walks with length three
  - what is A^3_{ii}?
  - if undirected, $Tr(A^3)/6$ gives the number of triangles
  - we compute number of triangles more effectively from eigenvalues of A as $\frac{1}{6}\sum_i \lambda_i^3$, since if λ is eigenvalue of A then λ^p is eigenvalue of A^p
  - real world graphs usually have a lot of triangles, e.g. friends of friends are friends



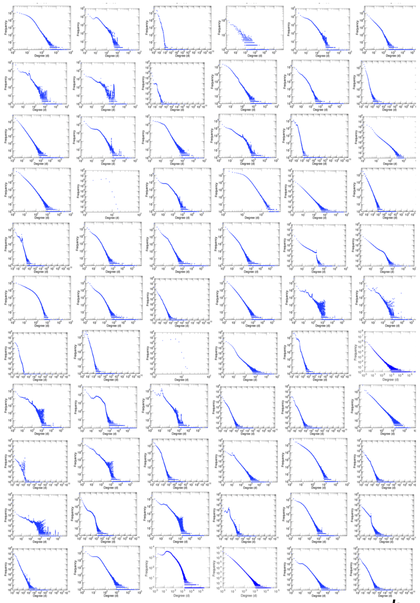
Degree distribution



$$A = \begin{array}{cc} \begin{bmatrix} A_{11}, & A_{12}, & \cdots, & A_{1N} \\ \vdots & \vdots & \ddots & \vdots \\ A_{N1}, & A_{N2}, & \cdots, & A_{NN} \end{bmatrix} & \text{inlinks} \\ \text{outlinks} & \end{array}$$

- marginals of A are called **degree**
  - $d_i = \sum_j A_{ij}$
  - if directed, ($A_{ij} = 1$ there is an edge from node j to i) we have
    - column-wise and row-wise marginals as indegree and out degree of nodes
    - $d_i^{in} = \sum_j A_{ij}$, and $d_i^{out} = \sum_j A_{ji}$
- $\sum_i \sum_j A_{ij}$
  - total number of edges (if directed), or twice that if undirected

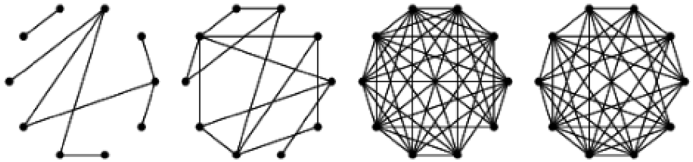
- **degree distribution:** how many nodes of degree k are in the graph
  - is often heavy tailed in real world networks (there are few nodes with very high degree & many with very small degree)
- degree distribution is plotted in log-log and a line could give a goof fit
  - $ln(p(d)) = -\alpha ln(d) + ln(c) \Rightarrow p(d) = cd^{-\alpha}$: powerlaw distribution
- often referred to as being **scale-free** since
  - $p(\lambda d) = \lambda^{-\alpha} c d^{-\alpha}$



example degree distributions

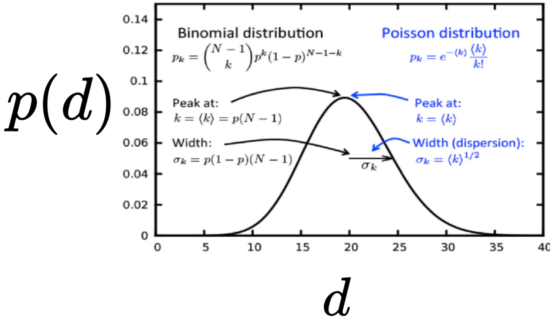
Real-world v.s. random graphs

Erdös-Rényi Model (ER) graphs

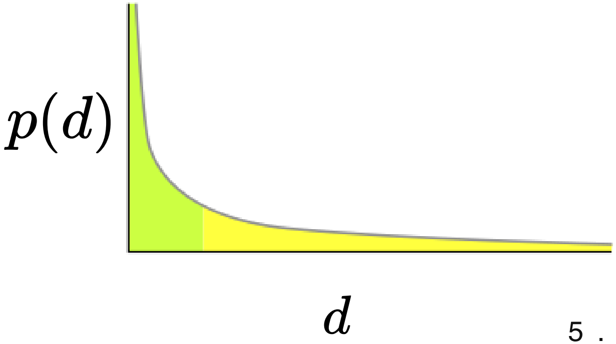


- basis of random graph theory
- simple model that results in small-world graphs
- parameters: ER(n, p) or ER(n, m)
  - n: number of nodes
  - p: probability of an edge between any two nodes
  - m: number of edges
- generation: all edges are equally likely so toss n(n-1)/2 coins

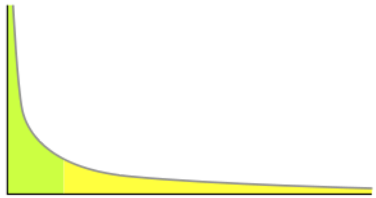
Degree distribution:



compare with real world graphs which have a heavy tail



Powerlaws



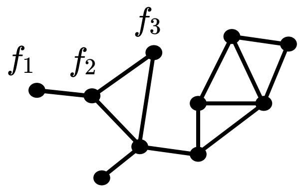
a common distribution

- Income follow a Pareto distribution
  - few individuals earned most of the money & majority earned small amounts
  - in the US 1% of the population earns a disproportionate 15% of the total US income
  - 80/20 rule (Pareto principle): a general rule of thumb
    - e.g. 20 percent of the code has 80 percent of the errors
- Zipf's law
  - distribution of words ranked by their frequency in a random text corpus is approximated by a power-law distribution
  - the second item occurs approximately 1/2 as often as the first, and the third item 1/3 as often as the first, and so on

preferential attachment which results in scale-free graphs

- node is connected to existing nodes with $p(i) \propto d_i$

Spectral clustering



consider function f that maps vertices to a value

$$f = (f_1, f_2, \dots f_N) \in \mathbb{R}^N \Rightarrow f^\top L f = \frac{1}{2}\sum_{ij} A_{ij}(f_i - f_j)^2$$

measures how much the value of f is smooth over edges, i.e. the difference of values for connecting nodes

How to cluster? Find f that give smoothest results, i.e, minimizes this

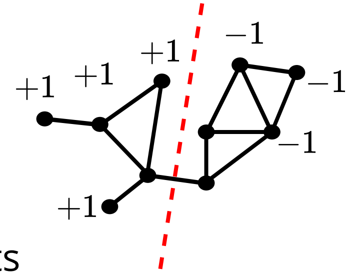
$$f_i \in \{+1, -1\} \text{ and } \sum_i f_i = 0$$

relaxed

Courant Fisher Minmax Theorem

$$f_i \in \mathbb{R} \text{ and } \sum_i f_i^2 = N \Rightarrow \min f^\top L f = N\lambda_1$$

- second smallest eigenvalue ⇒ sparsest cut
- signs of corresponding **eigenvector** ⇒ cluster assignments



more than 2 clusters? use k-means on top k eigenvectors (each node is represented with k features)

read more here

Clustering Graphs

Better choices for graphs:

- modularity optimization
  - number of links between them is more than chance, examples: FastModularity, **Louvain**

the best default

- random walk based
-   - Within them a random walk is more likely to trap, e.g. Walktrap
- compression based
  - Coding gives efficient compression of any random walk, e.g. Infomap
- centroid based
  - follow their closest leader e.g. TopLeader

Clustering Graphs

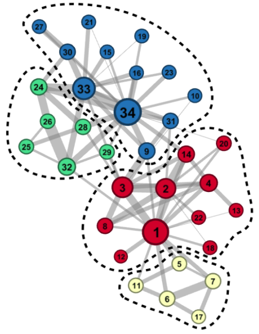
- Modularity optimization
  - number of links between them is more than chance
  - e_{ij}: fraction of edges between cluster i and j, and $a_i = \sum_j e_{ij}$

$$Q = \sum_i (e_{ii} - a_i^2) = Tr(e) - ||e^2||_1$$

here $||e^2||_1 = \sum_{ij} e_{ij}^2$

example

$$e = \begin{bmatrix} 0.71 & 0.35 & 0.06 & 0. \\ 0.22 & 0.3 & 0.22 & 0. \\ 0.065 & 0.35 & 0.59 & 0.4 \\ 0. & 0. & 0.12 & 0.6 \end{bmatrix}$$



optimize with an agglomerative hierarchical clustering

- merge two cluster that give the highest gain in Q

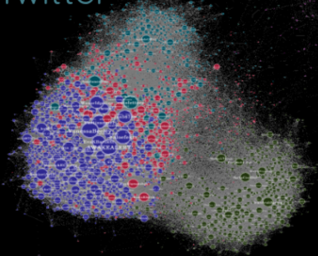
$$\Delta Q = 2(e_{ij} - a_i a_j)$$

FastModularity

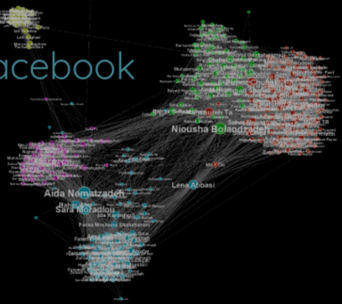
uses this with heap based data structure ⇒ O(m log n)

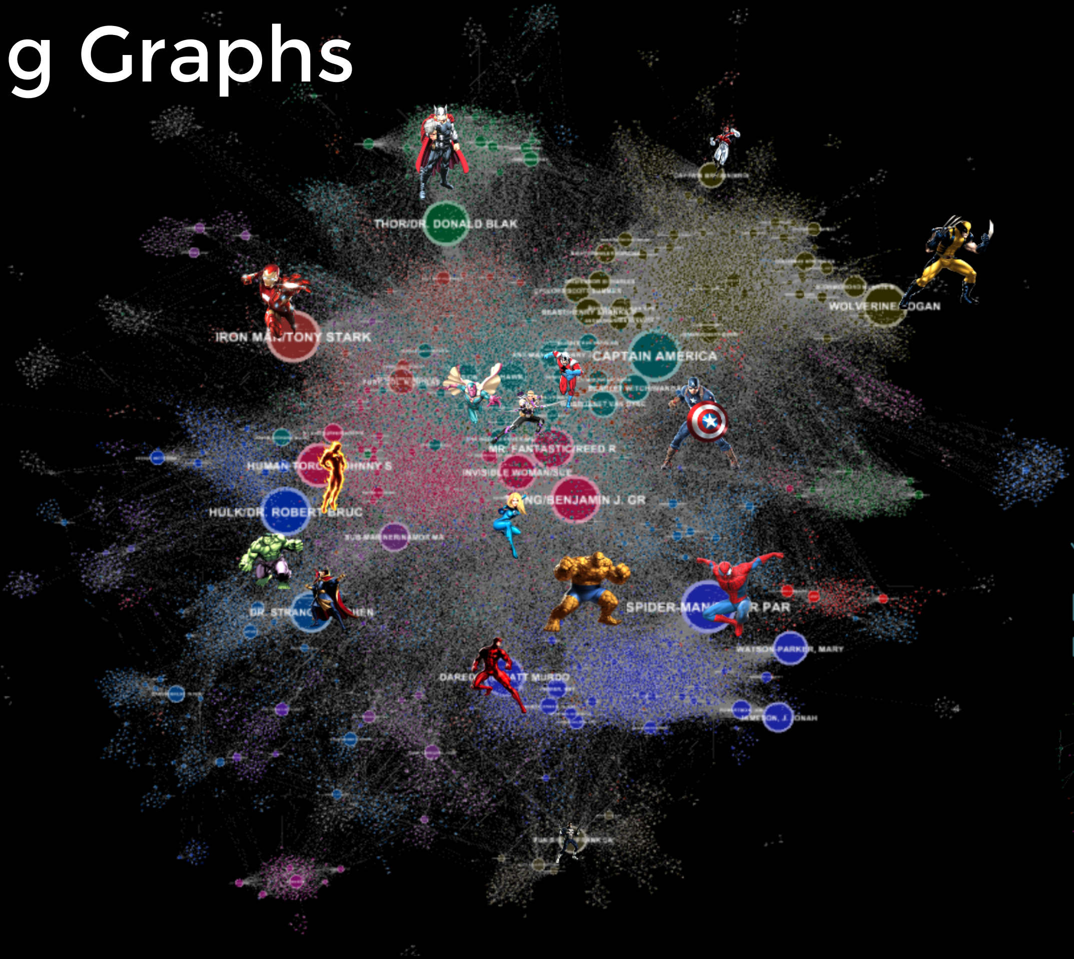
Clustering Graphs

Twitter



Facebook





C. elegans
neural network

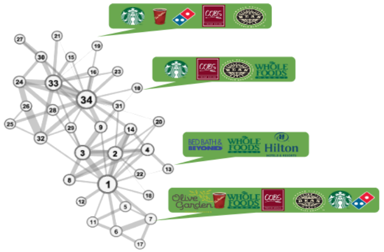
Yeast protein
protein interaction
networks

Attributed Graphs

Individual characteristics or activity (attributes) & relations (graph)

characteristics

age, occupation, salary, sex, etc.



activity & interest

check-ins, page-likes, group memberships, movies

Interplay between attributes and relations, a positive feedback loop derived by two social theories:

- **social selection**
  - similarity of individuals' characteristics motivates them to form relations
- **social influence**
  - characteristics of individuals may be affected by the characteristics of their relations
    - your neighbours' attributes can reveal yours

inductive bias: homophily

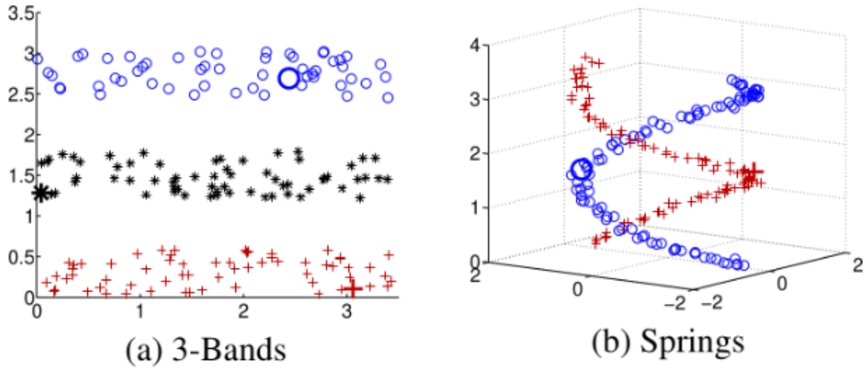
birds of the same feather flock together

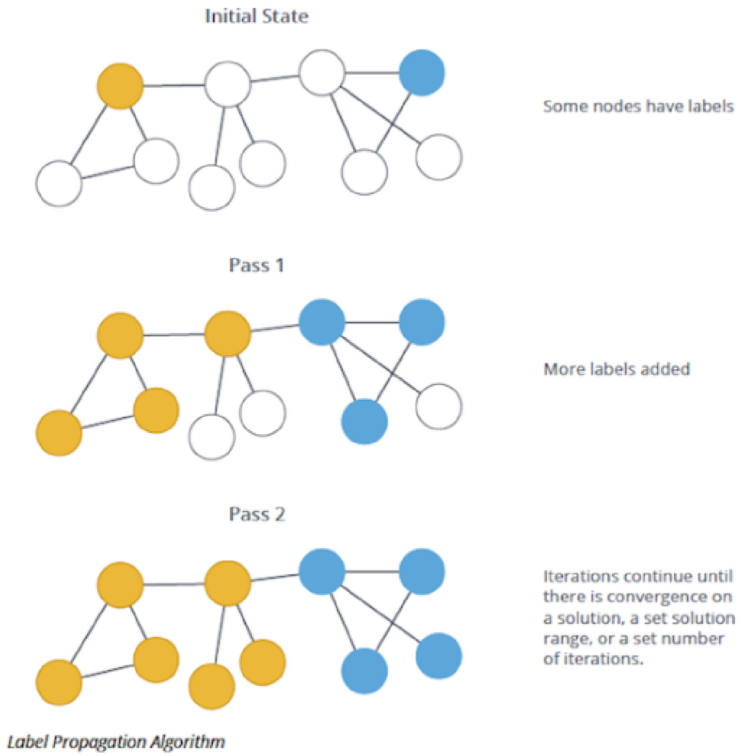
Node classification

Label Propagation Algorithm

label = mean (scalar) & mode (categorical) of your neighbors

proposed for semi-supervised classification of iid data by defining a fully connected distance graph



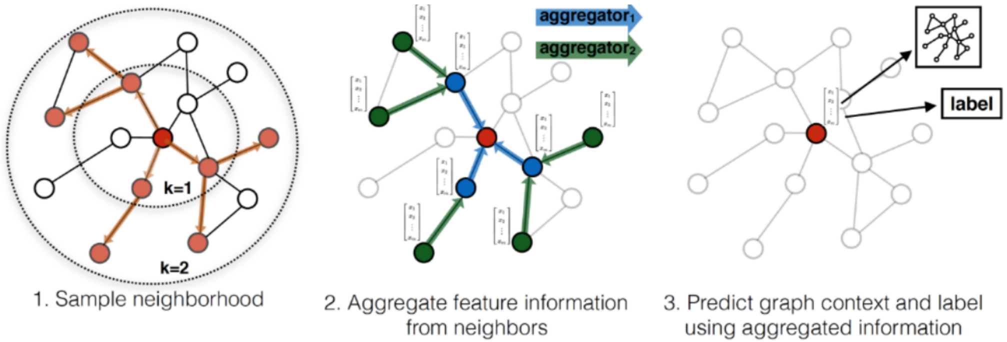


Node classification

- Unsupervised learning
  - clustering, only graph is given, classes/clusters are not predefined
- Supervised learning
  - classifying, input is graph and labels on all nodes
    - You mask some nodes (labels and their connections) for training [inductive]
    - You mask some nodes (only labels) for training [transductive]
- Semi-supervised learning
  - input is graph and labels on some nodes
  - You mask some node labels for training (seeing the whole graph: transductive)
- Active learning
  - Input is graph and a budget that determines how many nodes you can query for labels
  - labels come in sequence and can be queried based on the current set

Semi-Supervised Node classification

- classic methods
  - label propagation & belief propagation
- recent end-to-end methods (feature smoothing)
  - GCN and variants, which use a classification loss
- embedding based
  - unsupervised embedding extraction (e.g. node2vec) then apply a classifier



read more here

Summary

- graphs are everywhere
- real world graphs have special patterns
- graphs are represented with matrices
- graph clustering partitions the nodes in a graph
- node classification labels the node for which label is missing
- there are other tasks: link prediction, graph classification, ranking, etc.