EECE-6822 Machine Learning

Graph Neural Networks (GNNs) Learning on Networks

Cris Ababei
Dept. of Electrical and Computer Engineering
MARQUETTE UNIVERSITY
BE THE DIFFERENCE.

1

PART 1
GNNs

2

Graph Data

3

Graphs

- Represent a certain way we describe and capture relationships in data
- Particular kind of data structure that is nonlinear and abstract

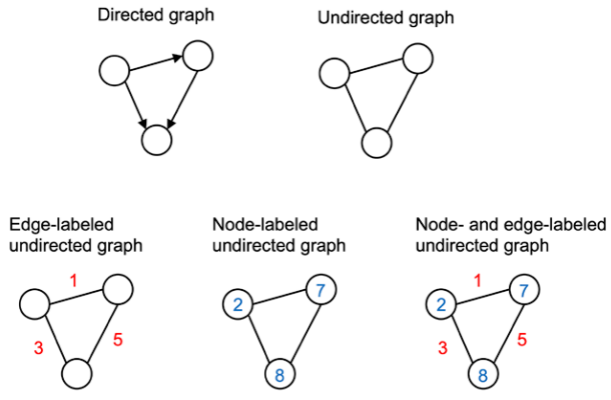


Figure 18.1: Common types of graphs

[***B3-Raschka***] Sebastian Raschka, Yuxi Liu, and Vahid Mirjalili, Learning with PyTorch and Scikit-Learn: Develop machine learning and deep learning models with Python, Packt Publishing, 2022.

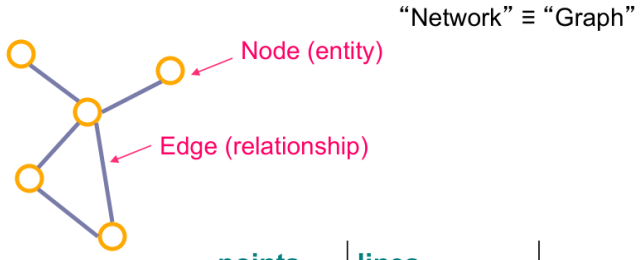
4

Why Graphs?

- Many real-world systems are **relational**, not Euclidean:
  - Molecules (atoms bonded)
  - Social networks (people connected)
  - Transportation networks
  - Knowledge graphs
  - Citation networks
  - Protein structures
  - Recommender systems
- A GNN can naturally capture **dependencies across neighbors**, similar to how CNNs capture local patterns in images.

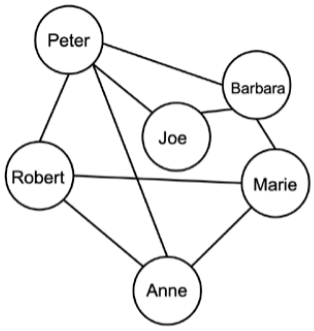
What are networks?

- Networks are collections of entities joined by relationships

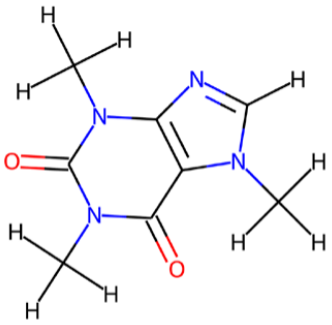


points	lines	
vertices	edges, arcs	math
nodes	links	computer science
sites	bonds	physics
actors	ties, relations	sociology

Examples - undirected



Friend graph



Molecular graph of caffeine

Figure 18.2: Two examples of undirected graphs

7

7

Examples - directed

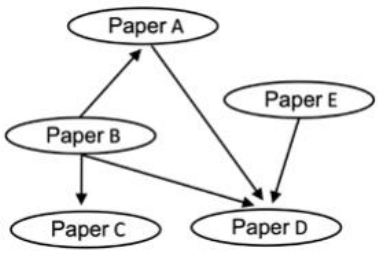
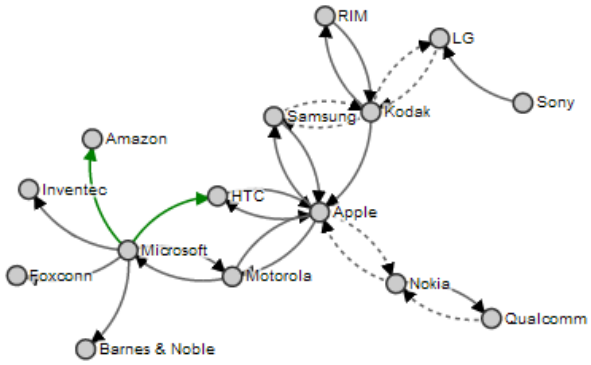


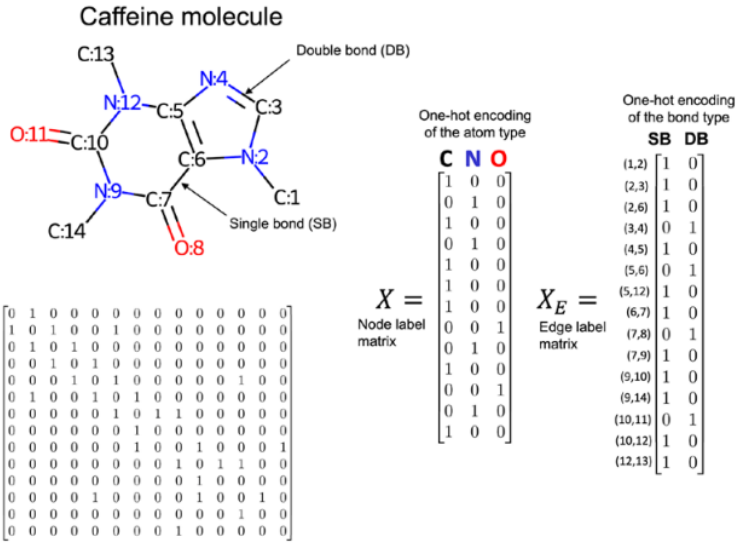
Figure 18.3: An example of a directed graph



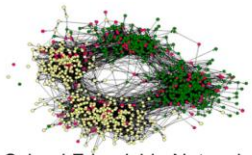
8

8

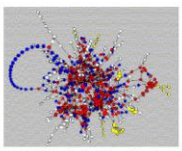
Representing molecules as undirected graphs



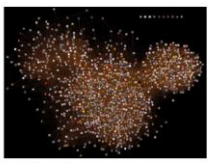
Example Relational Networks



School Friendship Network
(from Moody 2001)

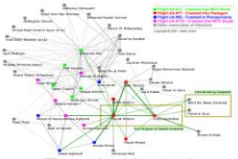


Yeast Metabolic Network
(from https://www.nd.edu/~networks/cell/)

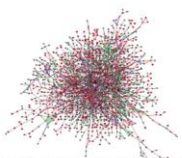


Flikr Social Network
from http://www.flickr.com/photos/gustavog/sets/164006/)

Campaign Contributions from Oil Companies
(from http://oilmoney.priceofoil.org/)

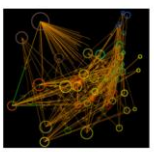


Terrorist Network
(by Valdis Krebs, Orgnet.com)



Protein-Protein Interactions
(by Peter Uetz)

Genomic Associations
(from Snel et al., 2002)

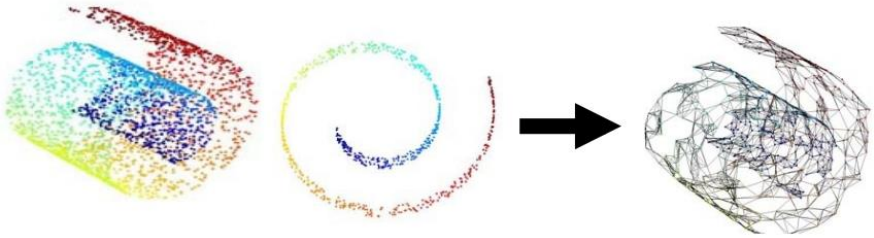


Seagrass Food Web
(generated at http://drjoe.biology.ecu.edu)

Creating a network from a surface

- Sample points from the surface
- Connect each point to the k closest points as measured by Euclidean distance

$$\text{dist}(\mathbf{x}_i, \mathbf{x}_j) = \|\mathbf{x}_i - \mathbf{x}_j\|_2$$



11

11

Creating a network from data

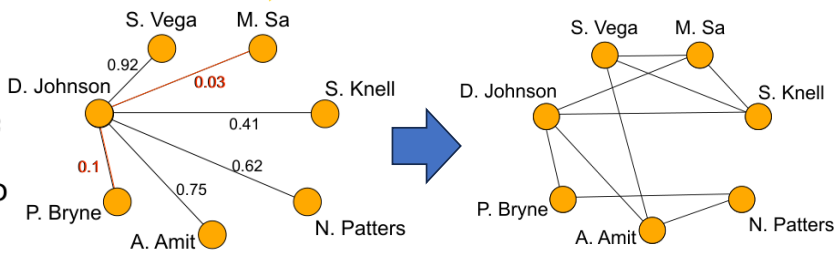
Medical Patients

Name	Age	Weight	Height	HR	SBP	DBP	SpO_2	...
D. Johnson	32	153	70	82	134	72	98%	...
S. Knell	47	169	65	130	169	93	99%	...
P. Bryne	42	128	61	102	129	77	98%	...
A. Amit	39	191	68	121	143	92	96%	...
...	...	...	...	...	...	...	...	...

1.) Measure the distance between pairs

$$\text{dist}(\mathbf{x}_i, \mathbf{x}_j) = \|\mathbf{x}_i - \mathbf{x}_j\|_2$$

2.) Connect each patient to its k nearest neighbors



12

12

Graph Convolutions

Key component for building GNNs

13

13

Recall from CNNs

- For CNNs, the **filter** can be viewed as a detector for a specific feature.
- This approach to feature detection is well-suited for images for several reasons, for instance, the following **priors** we can place on image data:
  1. **Shift-invariance:** We can still recognize a feature in an image regardless of where it is located (for example, after translation). A cat can be recognized as a cat whether it is in the top left, bottom right, or another part of an image.
  2. **Locality:** Nearby pixels are closely related.
  3. **Hierarchy:** Larger parts of an image can often be broken down into combinations of associated smaller parts. A cat has a head and legs; the head has eyes and a nose; the eyes have pupils and irises.

14

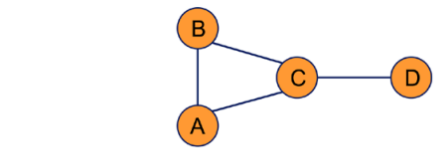
14

Graphs also have natural priors that justify a convolutional approach

- Both kinds of data, images and graphs, share the **locality prior:**
  - However, how we define locality differs. In images, the prior is on locality in 2D space, while with graphs, it is structural locality - a node that is one edge away is more likely to be related than a node five edges away.
- A strict prior for graph data is **permutation invariance:**
  - Ordering of the nodes does not affect the output.

15

15



Adjacency matrix 1:

	D	B	A	C
D	0	0	0	1
B	0	0	1	1
A	0	1	0	1
C	1	1	1	0

Adjacency matrix 2:

	A	C	D	B
A	0	1	0	1
C	1	0	1	1
D	0	1	0	0
B	1	1	0	0

Adjacency matrix 3:

	A	B	C	D
A	0	1	1	0
B	1	0	1	0
C	1	1	0	1
D	0	0	1	0

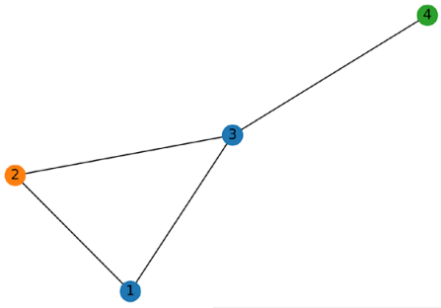
Figure 18.5: Different adjacency matrices representing the same graph

- Since the same graph can be represented by multiple adjacency matrices, any graph convolution needs to be permutation invariant
- While image convolutional operators are standardized, there are many different kinds of graph convolutions
- **Development of new graph convolutions is a very active area of research**

16

16

Implementing a basic graph convolution



n×n **adjacency** matrix A

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

n×f_{in} **node feature** matrix X

$$X = \begin{matrix} G & B & O \\ \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \end{matrix}$$

Installing NetworkX

NetworkX is a handy Python library for manipulating and visualizing graphs. It can be installed via `pip`:

```
pip install networkx
```

We used version 2.6.2 to create the graph visualizations in this chapter. For more information, please visit the official website at `https://networkx.org`.

17

17

Graph convolutions

With graph convolutions, we can interpret each row of X as being an embedding of the information that is stored at the node corresponding to that row. Graph convolutions update the embeddings at each node based on the embeddings of their neighbors and themselves. For our example implementation, the graph convolution will take the following form:

$$\boldsymbol{x}_i' = \boldsymbol{x}_i \boldsymbol{W}_1 + \sum_{j \in N(i)} \boldsymbol{x}_j \boldsymbol{W}_2 + b$$

Here, $\boldsymbol{x}_i'$ is the updated embedding for node i; $\boldsymbol{W}_1$ and $\boldsymbol{W}_2$ are $f_{in} \times f_{out}$ matrices of learnable filter weights; and b is a learnable bias vector of length f_{out}.

The two weight matrices $\boldsymbol{W}_1$ and $\boldsymbol{W}_2$ can be considered filter banks, where each column is an individual filter. Note that this filter design is most effective when the locality prior on graph data holds. If a value at a node is highly correlated with the value at another node many edges away, a single convolution will not capture that relationship. Stacking convolutions will capture more distant relationships, as illustrated in *Figure 18.7* (we set the bias to zero for simplicity):

18

18

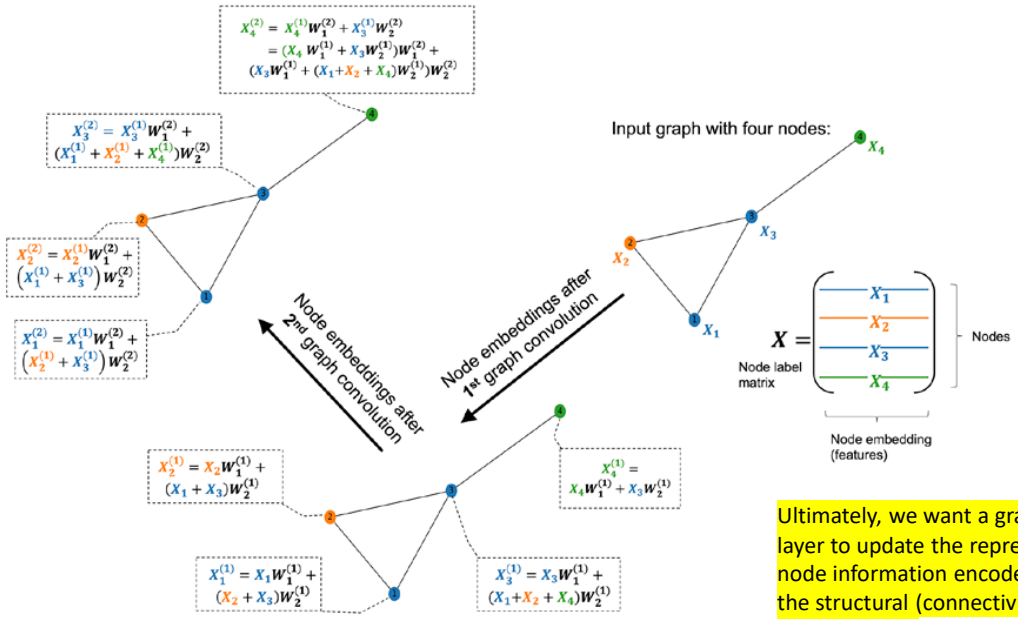


Figure 18.7: Capturing relationships from a graph

Ultimately, we want a graph convolutional layer to update the representation of the node information encoded in X by utilizing the structural (connectivity) information provided by A.

19

19

Message-passing framework

- Framework presented in: Neural Message Passing for Quantum Chemistry by Justin Gilmer and colleagues, 2017, https://arxiv.org/abs/1704.0121

In this message-passing framework, each node in the graph has an associated hidden state $h_i^{(t)}$, where i is the node's index at time step t. The initial value $h_i^{(0)}$ is defined as $\boldsymbol{X}_i$, which is the row of $\boldsymbol{X}$ associated with node i.

Each graph convolution can be split into a message-passing phase and a node update phase. Let $N(i)$ be the neighbors of node i. For undirected graphs, $N(i)$ is the set of nodes that share an edge with node i. For directed graphs, $N(i)$ is the set of nodes that have an edge whose endpoint is node i. The message-passing phase can be formulated as follows:

$$m_i = \sum_{j \in N(i)} M_t\left(h_i^{(t)}, h_j^{(t)}, e_{ij}\right)$$

Here, M_t is a message function. In our example layer, we define this message function as $M_t = h_j^{(t)} W_2$. The node update phase with the update function U_t is $h_i^{(t+1)} = U_t\left(h_i^{(t)}, m_i\right)$. In our example layer, this update is $h_i^{(t+1)} = h_i^{(t)} W_1 + m_i + b$.

20

20

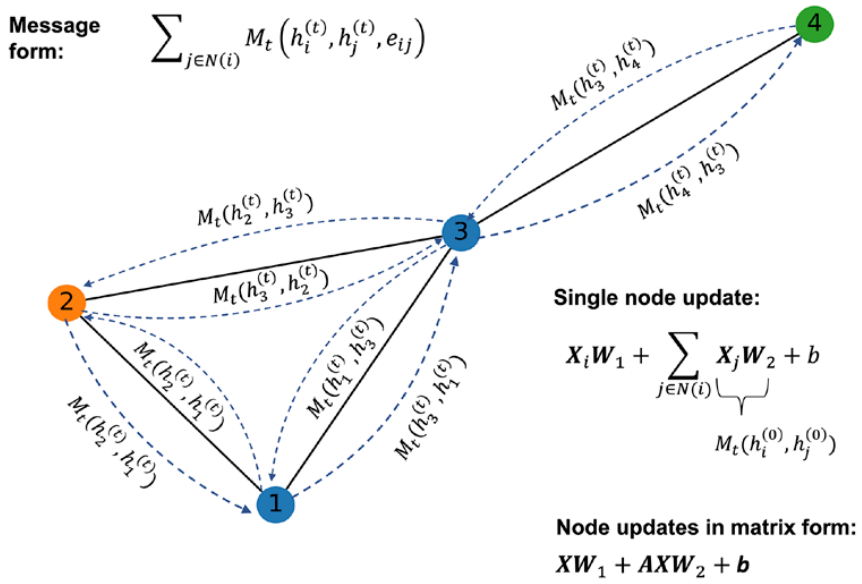


Figure 18.8: The convolutions implemented on the graph and the message form

21

21

Implementing a GNN in PyTorch from scratch

22

22

What Is a Graph Neural Network?

A **Graph Neural Network (GNN)** is a neural architecture designed to operate on **graph-structured data**.
Graphs consist of:

- **Nodes (vertices)**
- **Edges (relationships)**
- Optional: node features, edge features, global graph features

Formally, a graph $G = (V, E)$ where:

- V = set of nodes, $|V| = N$
- E = set of edges
- $X \in \mathbb{R}^{N \times d}$ = node feature matrix
- $A \in \mathbb{R}^{N \times N}$ = adjacency matrix

23

23

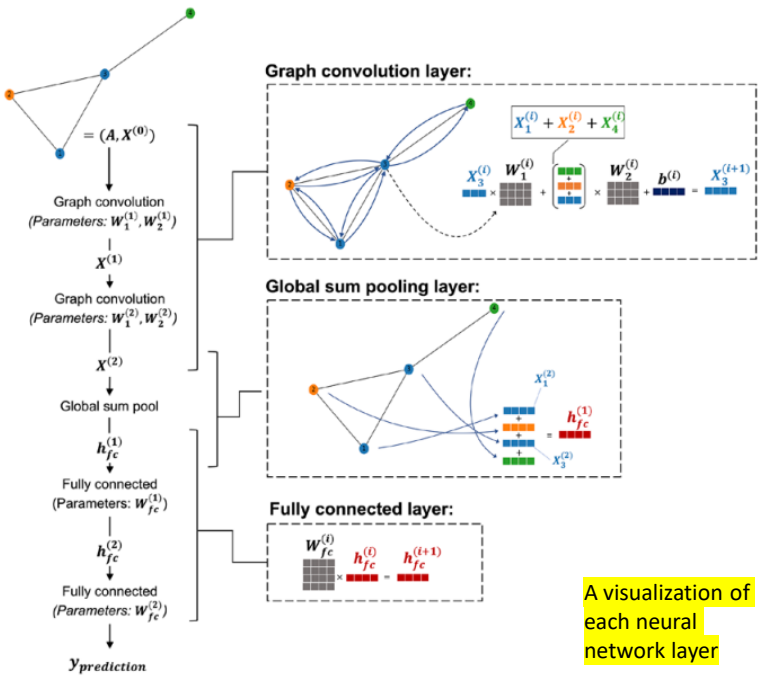
Defining the **NodeNetwork** model

```
import networkx as nx
import torch
from torch.nn.parameter import Parameter
import numpy as np
import math
import torch.nn.functional as F

class NodeNetwork(torch.nn.Module):
    def __init__(self, input_features):
        super().__init__()
        self.conv_1 = BasicGraphConvolutionLayer (
            input_features, 32)
        self.conv_2 = BasicGraphConvolutionLayer(32, 32)
        self.fc_1 = torch.nn.Linear(32, 16)
        self.out_layer = torch.nn.Linear(16, 2)

    def forward(self, X, A, batch_mat):
        x = F.relu(self.conv_1(X, A))
        x = F.relu(self.conv_2(x, A))
        output = global_sum_pool(x, batch_mat)
        output = self.fc_1(output)
        output = self.out_layer(output)
        return F.softmax(output, dim=1)
```

24

24

Structure of the network



A visualization of each neural network layer

25

Graph convolution layers and global pooling

- See **S.Raschka**'s book for implementation details in Python of GNN from scratch:
  - Coding the NodeNetwork's graph convolution layer
  - Adding a global pooling layer to deal with varying graph sizes
  - Preparing the DataLoader
  - Using the NodeNetwork to make predictions
- Also for:
  - Implementing a GNN using the PyTorch Geometric library

26

26

PART 2

Code Time!

Code Time

- See demonstration and discussion in class.
- See also links in the “Code Examples” for this lecture assignment.

Conclusion
Takeaways

Summary

A **Graph Neural Network**:

- processes **non-Euclidean, relational data**
- aggregates neighbor information
- updates node embeddings layer-by-layer
- learns representations for nodes, edges, or whole graphs

Core equation:

$$h_i^{(k+1)} = \text{UPDATE}\big(h_i^{(k)},\ \text{AGGREGATE}(\{h_j^{(k)} : j \in \mathcal{N}(i)\})\big)$$

Where:

- $h_i^{(k)}$ = embedding of node i at layer k
- $\mathcal{N}(i)$ = neighbors of i
- AGGREGATE = mean, sum, max, attention, etc.
- UPDATE = MLP, GRU, etc.

References and Credits

Many of the teaching materials for this course have been adapted from various sources. We are very grateful and thank the following professors, researchers, and practitioners for sharing their teaching materials (in no particular order):

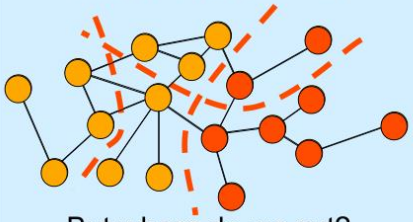
- Yaser S. Abu-Mostafa, Malik Magdon-Ismail and Hsuan-Tien Lin. https://amlbook.com/slides.html
- Ethem Alpaydin. https://www.cmpe.boun.edu.tr/~ethem/i2ml3e/
- Natasha Jaques. https://courses.cs.washington.edu/courses/cse446/25sp/
- Lyle Ungar. https://alliance.seas.upenn.edu/~cis520/dynamic/2022/wiki/index.php?n=Lectures.Lectures
- Aurelien Geron. https://github.com/ageron/handson-ml3
- Sebastian Raschka. https://github.com/rasbt/machine-learning-book
- Trevor Hastie. https://www.statlearning.com/resources-python
- Andrew Ng. https://www.youtube.com/playlist?list=PLoROMvodv4rMiGQp3WXShtMGgzqpfVfbU
- Richard Povineli. https://www.richard.povinelli.org/teaching
- … and many others.

31

Appendix A: Graph Partitioning

Goal: Partition the graph into multiple groups (*clusters*)

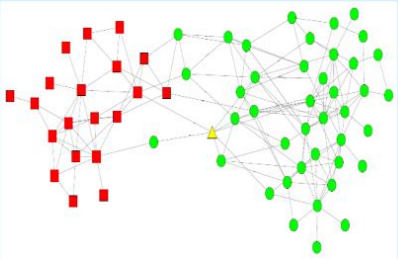
Find the two clusters



But where do we cut?

Identify the different parts of the rabbit

Social network of 62 dolphins
(Lusseau et al., 2003)



Predict how the community will split when △ departs

32

Spectral Clustering

- Algorithms that cluster points using eigenvectors of matrices derived from the data
- Obtain data representation in the low-dimensional space that can be easily clustered
- Variety of methods that use the eigenvectors differently

33

33

Spectral Clustering

- Empirically very successful
- Authors disagree:
  - Which eigenvectors to use
  - How to derive clusters from these eigenvectors
- Two general methods

34

34

Definitions

- n x n Adjacency matrix A.
  - A(i,j) = weight on edge from *i* to *j*
  - If the graph is undirected A(i,j)=A(j,i), i.e. A is symmetric
- n x n Transition matrix P.
  - P is row stochastic
  - P(i,j) = probability of stepping on node j from node i
    = $A(i,j)/\sum_i A(i,j)$
- n x n Laplacian Matrix L.
  - $L(i,j)=\sum_i A(i,j)-A(i,j)$
  - Symmetric positive semi-definite for undirected graphs
  - Singular

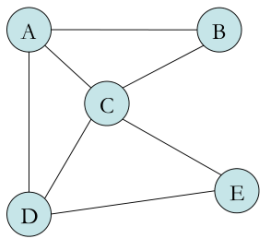
35

35

Definitions

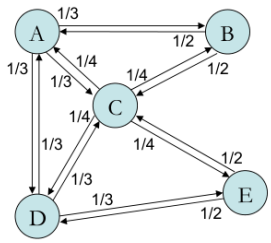
Adjacency Matrix

$$A=\begin{bmatrix} & 1 & 1 & 1 & \\ 1 & & 1 & & \\ 1 & 1 & & 1 & 1 \\ 1 & & 1 & & 1 \\ & & 1 & 1 & \end{bmatrix}$$



Transition Matrix

$$P=\begin{bmatrix} & 1/3 & 1/3 & 1/3 & \\ 1/2 & & 1/2 & & \\ 1/4 & 1/4 & & 1/4 & 1/4 \\ 1/3 & & 1/3 & & 1/3 \\ & & 1/2 & 1/2 & \end{bmatrix}$$



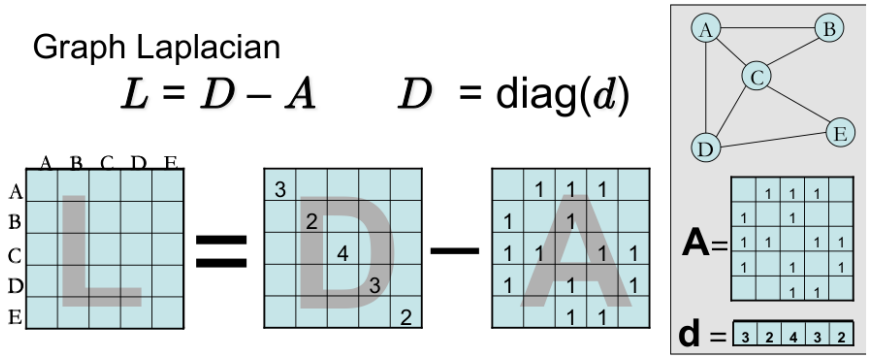
36

36

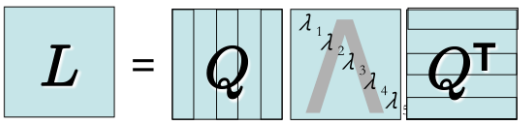
Spectral Graph Analysis

Graph Laplacian

$$L = D - A \qquad D = \text{diag}(d)$$



Take the *eigendecomposition* of L



37

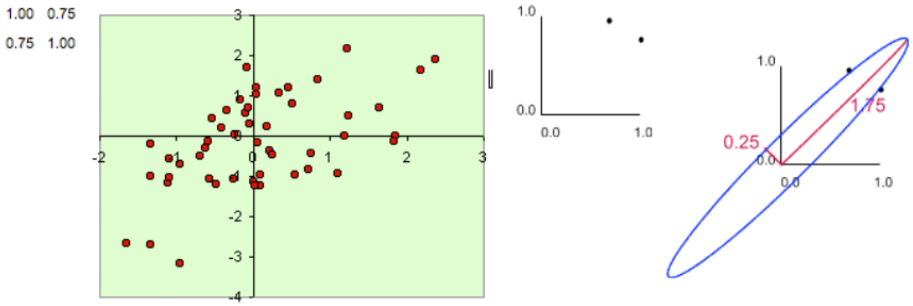
Eigenvectors

- Intuitive definition: An eigenvector is a direction for a matrix
- An eigenvector of an *n x n* matrix A is a vector such that $Av = \lambda v$, where *v* is the eigenvector and λ is the corresponding eigenvalue
  - Multiplying vector v by the scalar λ effectively stretches or shrinks the vector
- An *n x n* matrix should have *n* linearly independent eigenvectors

38

Eigenvectors Illustrated

- Consider an elliptical data cloud. The eigenvectors are then the major and minor axes of the ellipse



Spectral Graph Analysis

$$L = Q \Lambda Q^T$$

Q: $q_1\ q_2\ q_3\ q_4\ q_5$

Λ: $\lambda_1\ \lambda_2\ \lambda_3\ \lambda_4\ \lambda_5$

Q^T: q_1^T; q_2^T; q_3^T; q_4^T; q_5^T

Eigenvector q_1 is constant

Eigenvalue $\lambda_1 = 0$

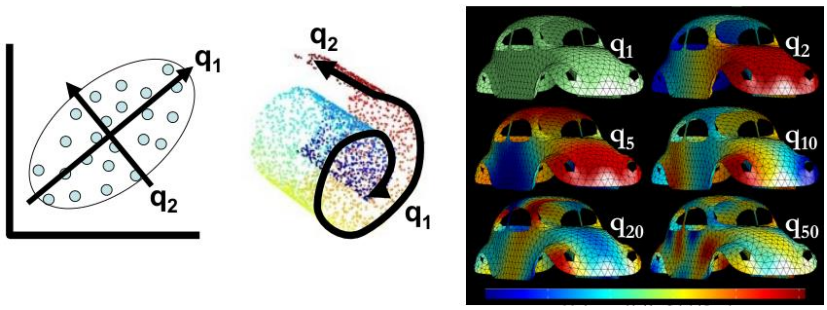
L				
3	-1	-1	-1	
-1	2	-1		
-1	-1	4	-1	-1
-1		-1	3	-1
		-1	-1	2

Q	q_1	q_2	q_3	q_4	q_5
A	0.45	-0.27	-0.5	-0.65	0.22
B	0.45	-0.65	0.5	0.27	0.22
C	0.45	-0.00	0.00	0.00	-0.89
D	0.45	0.27	-0.5	0.65	0.22
E	0.45	0.65	0.5	-0.27	0.22

Λ	1	2	3	4	5
1	0.00	0	0	0	0
2	0	1.59	0	0	0
3	0	0	3.00	0	0
4	0	0	0	4.41	0
5	0	0	0	0	5.00

Spectral Graph Analysis

$$L = Q \Lambda Q^T$$



41

41

Method #1

- Partition using only one eigenvector at a time
- Use procedure recursively
- Example: Image Segmentation
  - Uses 2nd (smallest) eigenvector to define optimal cut
  - Recursively generates two clusters with each cut

42

42

Method #2

- Use k eigenvectors (k chosen by user)
- Directly compute k-way partitioning
- Experimentally has been seen to be "better"

43

43

Spectral Clustering Algorithm (by Ng, Jordan, and Weiss)

- Given a set of points $\mathrm{X}=\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$
- Form the affinity matrix

$$A_{ij} = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|_2^2}{2\sigma^2}\right) \quad \forall i \neq j \quad A_{ii} = 0$$

- Define diagonal matrix $D_{ii} = \sum_k A_{ik}$
- Form the matrix $\mathbf{L} = \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$
- Stack the k largest eigenvectors of $\mathbf{L}$ to form the columns of the new matrix: $\mathbf{E} = [\mathbf{e}_1\ \mathbf{e}_2\ \ldots\ \mathbf{e}_k]$
- Normalize each of $\mathbf{E}$'s rows to have unit length
- Cluster rows of $\mathbf{E}$ into k clusters using K-means

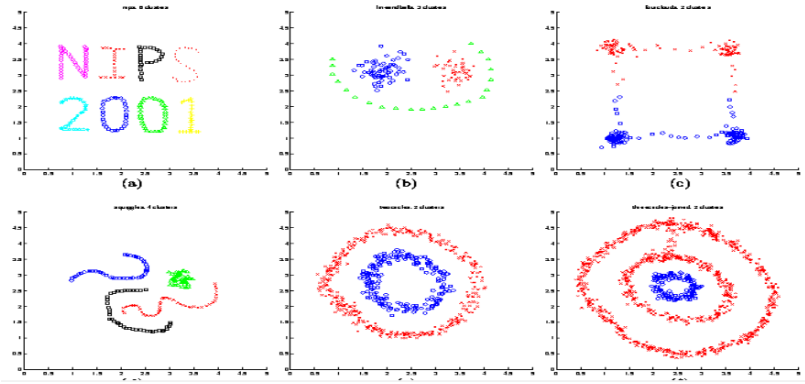
44

44

Why?

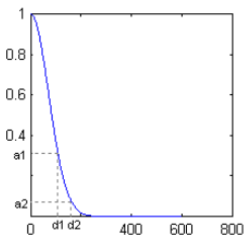
Q: If we eventually use K-means, why not just apply K-means to the original data?

A: This method allows us to cluster non-convex regions

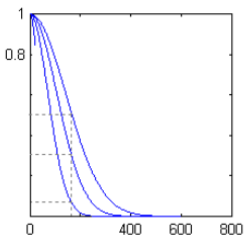


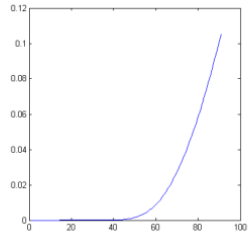
Nature of the Affinity Matrix

$$A_{ij} = e^{-(s_i - s_j)^2 / 2\sigma^2} \quad i \neq j \qquad A_{ii} = 0$$



"closer" vertices will get larger weight





Weight as a function of σ

User's Prerogative

- Choice of k, the number of clusters
- Choice of scaling factor
  - Realistically, search over σ^2 and pick value that gives the tightest clusters
- Choice of clustering method

47

47

(Some) References

- Alpert et al. Spectral partitioning with multiple eigenvectors
- Brand & Huang. A unifying theorem for spectral embedding and clustering
- Belkin & Niyogi. Laplasian maps for dimensionality reduction and data representation
- Blatt et al. Data clustering using a model granular magnet
- Buhmann. Data clustering and learning
- Fowlkes et al. Spectral grouping using the Nystrom method
- Meila & Shi. A random walks view of spectral segmentation
- Ng et al. On Spectral clustering: analysis and algorithm
- Shi & Malik. Normalized cuts and image segmentation
- Weiss et al. Segmentation using eigenvectors: a unifying view

48

48

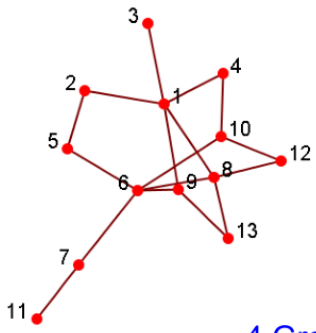
Community Structures
Community Detection

- A community is a set of nodes between which the interactions are (relatively) frequent
  a.k.a. group, subgroup, module, cluster
- Community detection
  a.k.a. grouping, clustering, finding cohesive subgroups
  - Given: a social network
  - Output: community membership of (some) actors
- Applications
  - Understanding the interactions between people
  - Visualizing and navigating huge networks
  - Forming the basis for other tasks such as data mining

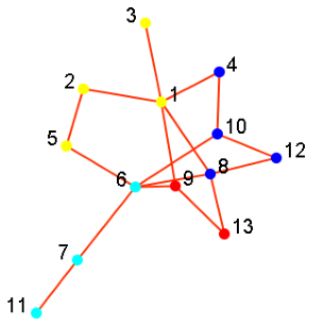
49

49

Visualization after Grouping



4 Groups:
{1,2,3,5}
{4,8,10,12}
{6,7,11}
{9,13}



(Nodes colored by Community Membership)

50

50

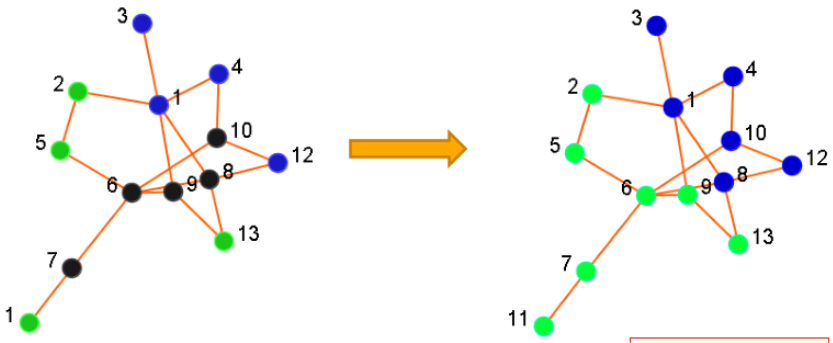
Classification

- User Preference or Behavior can be represented as class labels
  - Whether or not clicking on an ad
  - Whether or not interested in certain topics
  - Subscribed to certain political views
  - Like/Dislike a product
- Given
  - A social network
  - Labels of some actors in the network
- Output
  - Labels of remaining actors in the network

51

51

Visualization after Prediction



: Smoking
: Non-Smoking
: ? Unknown

Predictions
6: Non-Smoking
7: Non-Smoking
8: Smoking
9: Non-Smoking
10: Smoking

52

52

Link Prediction

- Given a social network, predict which nodes are likely to get connected
- Output a list of (ranked) pairs of nodes
- Example: Friend recommendation in Facebook

(2, 3)
(4, 12)
(5, 7)
(7, 13)

PRINCIPLES OF COMMUNITY DETECTION

Communities

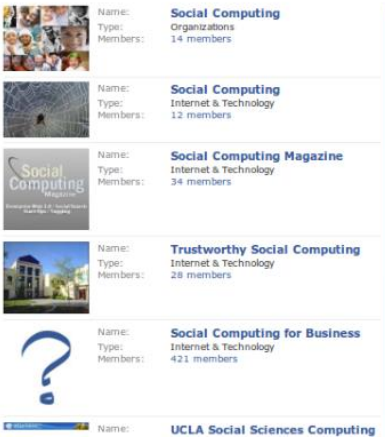
- Community: "subsets of actors among whom there are relatively strong, direct, intense, frequent or positive ties."
  -- Wasserman and Faust, *Social Network Analysis, Methods and Applications*
- Community is a set of actors interacting with each other *frequently*
- A set of people without interaction is NOT a community
  - e.g. people waiting for a bus at station but don't talk to each other

55

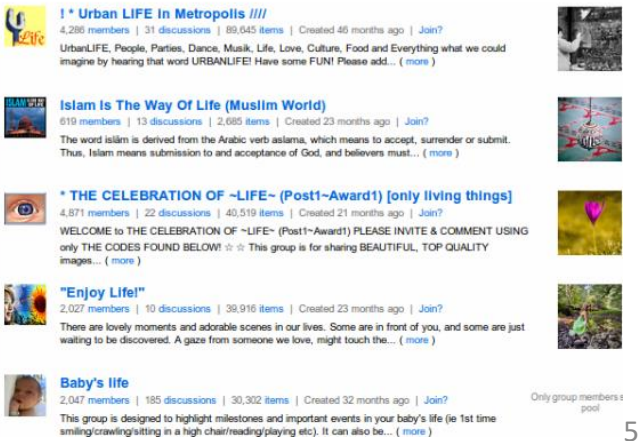
55

Example of Communities

Communities from Facebook



Communities from Flickr



56

56

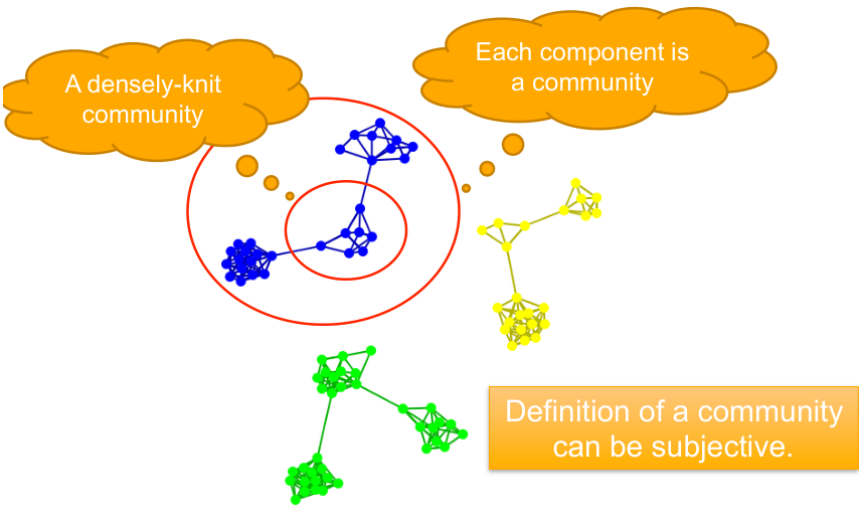
Community Detection

- Community Detection: "formalize the strong social groups based on the social network properties"
- Some social media sites allow people to join groups
  - Not all sites provide community platform
  - Not all people join groups
- Network interaction provides rich information about the relationship between users
  - Is it necessary to extract groups based on network topology?
  - Groups are *implicitly* formed
  - Can complement other kinds of information
  - Provide basic information for other tasks

57

57

Subjectivity of Community Definition



58

58

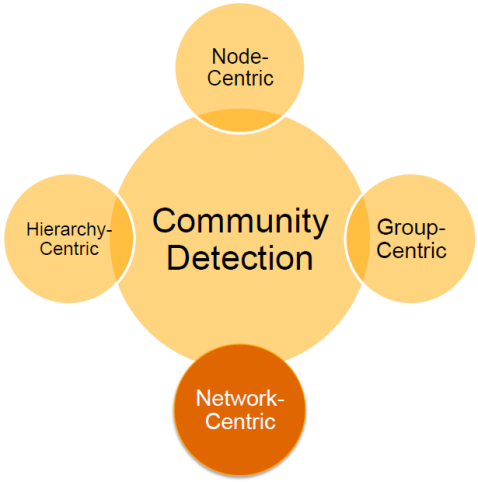
Taxonomy of Community Criteria

- Criteria vary depending on the tasks
- Roughly, community detection methods can be divided into 4 categories (not exclusive):
- Node-Centric Community
  - Each node in a group satisfies certain properties
- Group-Centric Community
  - Consider the connections within a group as a whole. The group has to satisfy certain properties without zooming into node-level
- Network-Centric Community
  - Partition the whole network into several disjoint sets
- Hierarchy-Centric Community
  - Construct a hierarchical structure of communities

59

59

Network-Centric Community Detection



60

60

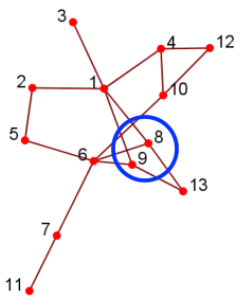
Network-Centric Community Detection

- To form a group, we need to consider the connections of the nodes globally.
- Goal: partition the network into disjoint sets
- Groups based on
  - Node Similarity
  - Latent Space Model
  - Block Model Approximation
  - Cut Minimization
  - Modularity Maximization

61

Node Similarity

- Node similarity is defined by how similar their interaction patterns are
- Two nodes are structurally equivalent if they connect to the same set of actors
  - e.g., nodes 8 and 9 are structurally equivalent
- Groups are defined over equivalent nodes
  - Too strict
  - Rarely occur in a large-scale
  - Relaxed equivalence class is difficult to compute
- In practice, use vector similarity
  - e.g., cosine similarity, Jaccard similarity



62

Vector Similarity Based on Adjacency Matrix

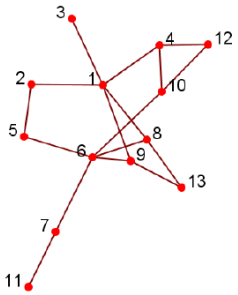
		1	2	3	4	5	6	7	8	9	10	11	12	13
a vector →	5		1				1							
structurally equivalent	8	1					1							1
	9	1					1							1

Cosine Similarity: $\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|}.$

$$sim(5,8) = \frac{1}{\sqrt{2} \times \sqrt{3}} = \frac{1}{\sqrt{6}}$$

Jaccard Similarity: $J(A,B) = \frac{|A \cap B|}{|A \cup B|}.$

$$J(5,8) = \frac{|\{6\}|}{|\{1,2,6,13\}|} = 1/4$$



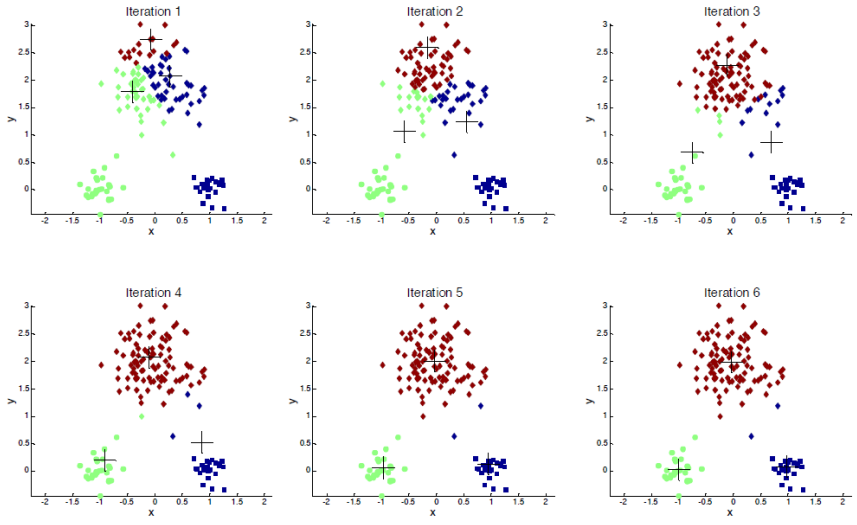
Clustering based on Node Similarity

- For practical use with huge networks:
  - Consider the connections as features
  - Use Cosine or Jaccard similarity to compute vertex similarity
  - Apply classical k-means clustering Algorithm

Algorithm 1 Basic K-means Algorithm.

1: Select K points as the initial centroids.
2: **repeat**
3: Form K clusters by assigning all points to the closest centroid.
4: Recompute the centroid of each cluster.
5: **until** The centroids don't change

Illustration of k-means clustering



Block-Model Approximation

Network Interaction Matrix Block Structure

- Objective: Minimize the difference between an interaction matrix and a block structure

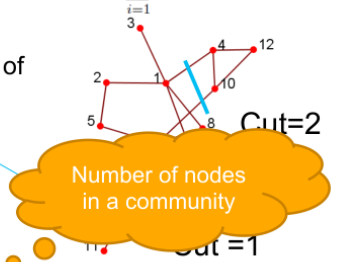
$$\min_{S,\Sigma} \|A - S\Sigma S^T\|_F$$
$$s.t. \quad S \in \{0,1\}^{n\times k}, \Sigma \in R^{k\times k} \text{ is diagonal}$$

S is a community indicator matrix

- Challenge: S is discrete, difficult to solve
- Relaxation: Allow S to be continuous satisfying $S^T S = I_k$
- Solution: the top eigenvectors of A
- Post-Processing: Apply k-means to S to find the partition

Cut-Minimization

- Between-group interactions should be infrequent
- Cut: number of edges between two sets of nodes
- Objective: minimize the cut $cut(C_1, C_2, \cdots, C_k) = \sum_{i=1}^{k} cut(C_i, \overline{C_i})$
  - Limitations: often find communities of only one node
  - Need to consider the group size



Number of nodes in a community

Number of within-group Interactions

- Two commonly-used variants:

$$\text{Ratio-cut}(C_1, C_2, \cdots, C_k) = \sum_{i=1}^{k} \frac{cut(C_i, \overline{C_i})}{|V_i|}$$

$$\text{Normalized-cut}(C_1, C_2, \cdots, C_k) = \sum_{i=1}^{k} \frac{cut(C_i, \overline{C_i})}{vol(V_i)}$$

Graph Laplacian

- Cut-minimization can be relaxed into the following min-trace problem

$$\min_{S \in R^{n \times k}} Tr(S^T L S) \quad s.t.\ S^T S = I$$

- L is the (normalized) Graph Laplacian

$$\begin{aligned} L &= D - A \\ \text{normalized-}L &= I - D^{-1/2} A D^{-1/2} \end{aligned} \qquad D = \begin{pmatrix} d_1 & 0 & \cdots & 0 \\ 0 & d_2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & d_n \end{pmatrix}$$

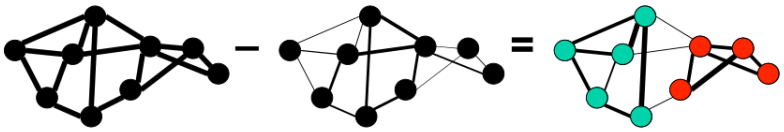
- Solution: S are the eigenvectors of L with smallest eigenvalues (except the first one)
- Post-Processing: apply k-means to S
  - a.k.a. Spectral Clustering

Graph Modularity

- Relational network given by $G = (V, A)$

  V: set of n vertices A : n x n adjacency matrix, m total edges

- Newman-Girvan (2006) graph modularity



Original A Null Model P Modularity (A-P)

– Measures the global community structure of G:

$$Q(C) = \frac{1}{2m}\sum_{i,j}(A_{ij} - P_{ij})\delta(C_i, C_j) \qquad P_{ij} = \frac{d_i d_j}{2m}$$

Kronecker delta

– Foundation for a large number of methods (Fortunato, 2010)

Modularity Maximization

- Modularity measures the group interactions compared with the expected random connections in the group
- In a network with m edges, for two nodes with degree d_i and d_j, expected random connections between them are

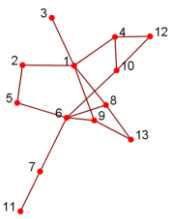
$$d_i d_j / 2m$$

- The interaction utility in a group:

$$\sum_{i\in C, j\in C} A_{ij} - d_i d_j / 2m$$

- To partition the group into multiple groups, we maximize

$$\frac{1}{2m}\sum_{C}\sum_{i\in C, j\in C} A_{ij} - d_i d_j / 2m$$



Expected Number of edges between 6 and 9 is
5*3/(2*17) = 15/34

Modularity Matrix

- The modularity maximization can also be formulated in matrix form

$$Q = \frac{1}{2m} Tr(S^T B S)$$

- B is the modularity matrix

$$B_{ij} = A_{ij} - d_i d_j / 2m$$

- Solution: top eigenvectors of the modularity matrix

71

71

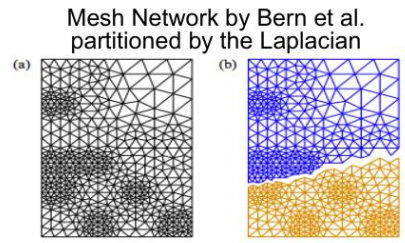
Properties of Modularity

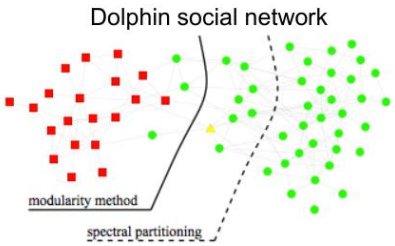
- Properties of modularity:
  - Between (-1, 1)
  - Modularity = 0 If all nodes are clustered into one group
  - Can automatically determine optimal number of clusters
- Resolution limit of modularity
  - Modularity maximization might return a community consisting multiple small modules

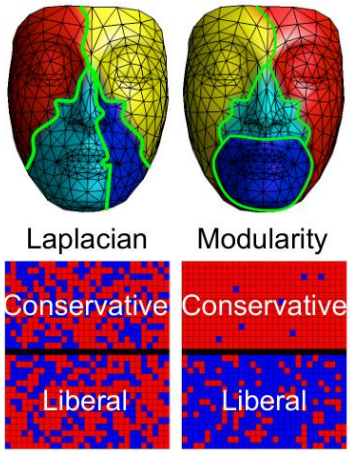
72

72

Graph Laplacian vs Graph Modularity







Political Blogs from 2004 U.S. Election,
data set from Adamic & Glance (2005)

73

73

Recap of Network-Centric Community

- Network-Centric Community Detection
  - Groups based on
    - Node Similarity
    - Latent Space Models
    - Cut Minimization
    - Block-Model Approximation
    - Modularity maximization
- Goal: Partition network nodes into several disjoint sets
- Limitation: Require the user to specify the number of communities beforehand

74

74