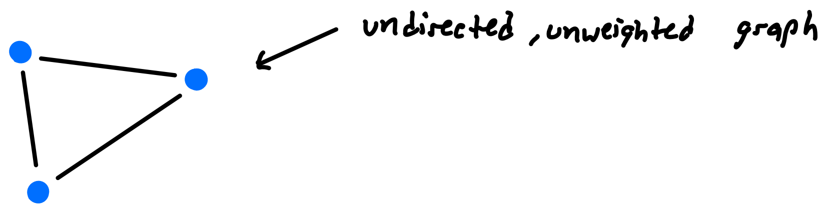
Community Detection

6/14/2022

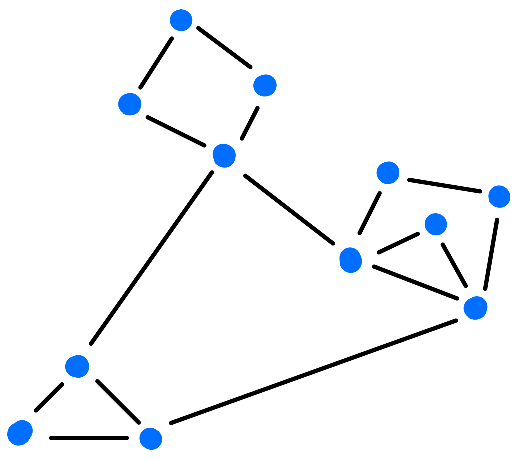
Agenda:
- Community detection definition
- modularity: tool for measuring how good a community split is
- community detection methods

Possible project idea: if you are interested in machine learning, a neural network (a brain for a computer) is just like a directed graph. It would be interesting to explore what insights Network Centrality can bring to machine learning.

Typically, community detection is done for undirected, unweighted graphs so that's what we'll focus on today but there are extensions to directed and weighted graphs as well



How many "communities" can you see in the following network?

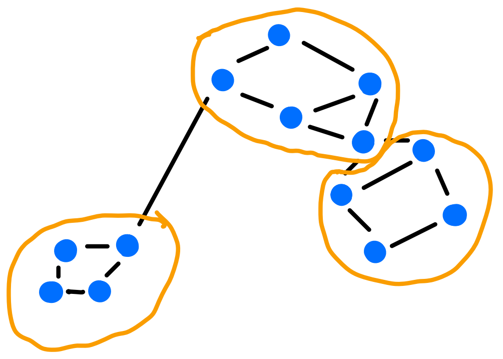


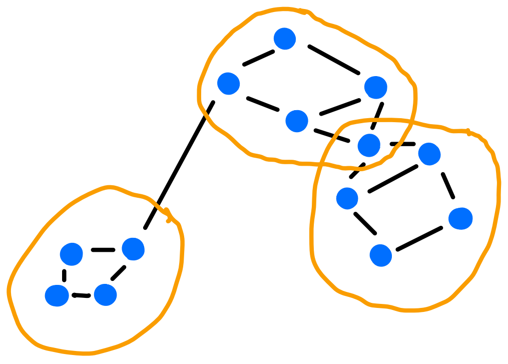
Community structure is an important feature of complex networks. Even in this small network above, we can already see community structure.

Why is community detection useful:
- personalizing ad experience
- friend suggestions on social media
- tweet analysis - each of the tweets are the nodes and use community detection if the tweet is saying something good or bad (sentiment analysis)

Community detection, also called graph partitioning, helps us reveal hidden relationships between nodes in a graph. Community detection is an assignment of nodes into communities (or groups).

Network Communities are groups of nodes such that the nodes inside the group are more connected to each other than to nodes outside the group.


Non-overlapping communities

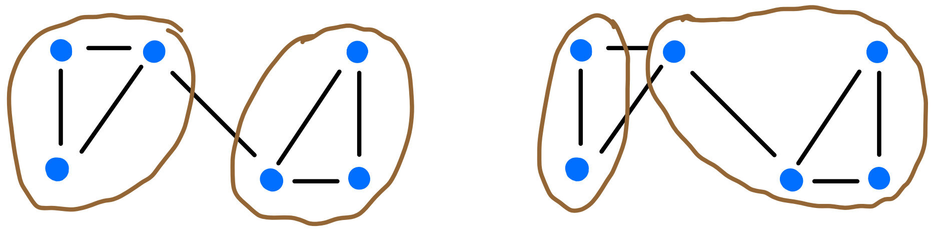

Overlapping communities

in different applications, you might need either one and there are different algorithms designed for both.

Many algorithm have been designed to do community detection, but we'll only explore 1 today.

First let's talk about modularity, a measure of the strength of a particular division of a network into communities.

ex)



Modularity is a way to mathematically quantify how good the split is. The left graph will have a larger modularity.

In words, the modularity is the number of edges falling within groups minus the expected number of edges in an equivalent network with edges placed at random.

In math, let's consider the problem of separating nodes into two communities.

Consider a network with n nodes

Let $s_i = \begin{cases} 1 & \text{if node } i \text{ belongs to community 1} \\ -1 & \text{if node } i \text{ belongs to community 2} \end{cases}$

$A_{ij} = \begin{cases} 1 & \text{if node } i \text{ is connected to node } j \\ 0 & \text{if node } i \text{ is not connected to node } j \end{cases}$

$\uparrow$ adjacency matrix

k_i = the degree of node i

$m = \frac{1}{2}\sum_{i=0}^{n-1} k_i$ the total number of edges in the network

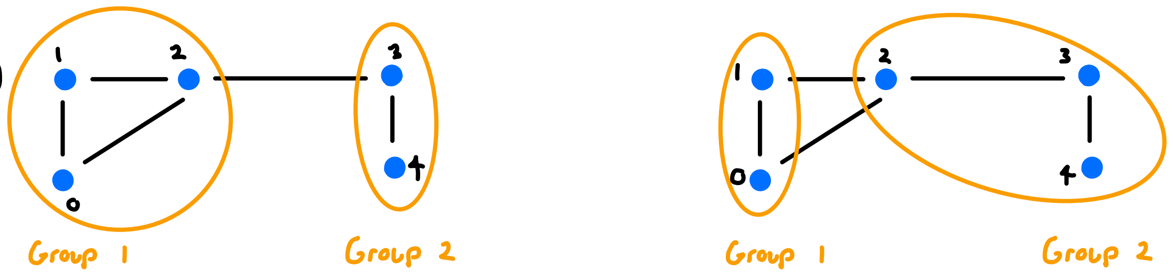
The expected number of edges between nodes i and j (while only knowing the degree of the nodes) if the edges are placed at random: $\frac{k_i k_j}{2m}$

The modularity Q, up to a constant, is given by the sum of $A_{ij} - \frac{k_i k_j}{2m}$ over all pairs of vertices i, j that fall in the same group.

Note: $\frac{1}{2}(s_i s_j + 1) = \begin{cases} 1 & \text{if } i \text{ and } j \text{ are in the same community} \\ 0 & \text{if } i \text{ and } j \text{ are not in the same community} \end{cases}$

$$\Rightarrow Q = \frac{1}{2m}\sum_{i=0}^{n-1}\sum_{j=0}^{n-1}\left(A_{ij} - \frac{k_i k_j}{2m}\right)\left(\frac{s_i s_j + 1}{2}\right) \longleftarrow \text{don't sum over terms where } i=j$$
$$= \frac{1}{2m}\sum_{i,j}\left(A_{ij} - \frac{k_i k_j}{2m}\right)\left(\frac{s_i s_j + 1}{2}\right)$$

ex)



$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}, \quad s = \begin{bmatrix} 1 \\ 1 \\ 1 \\ -1 \\ -1 \end{bmatrix}, \quad k = \begin{bmatrix} 2 \\ 2 \\ 3 \\ 2 \\ 1 \end{bmatrix}$$

$$Q = \frac{1}{2n}\left[\left(A_{01} - \frac{k_0 k_1}{2m}\right)\left(\frac{s_0 s_1 + 1}{2}\right) + \left(A_{02} - \frac{k_0 k_2}{2m}\right)\left(\frac{s_0 s_1 + 1}{2}\right) + \cdots\right.$$
$$= \frac{1}{2m}\left[2\cdot(\text{term } 0\to 1) + 2(\text{term } 0\to 2) + 2(\text{term } 1\to 2) + 2\cdot(\text{term } 3\to 4)\right]$$
$$= \frac{1}{2m}\left[2\cdot\left(A_{01} - \frac{k_0 k_1}{2m}\right)\left(\frac{s_0 s_1 + 1}{2}\right) + \quad\backslash\quad + \quad / \quad + \quad | \quad\right]$$
$$= \frac{1}{10}\left[2\cdot\left(1 - \frac{2\cdot 2}{10}\right)\cdot\left(\frac{1\cdot 1 + 1}{2}\right) + \quad\backslash\quad + \quad\backslash\quad + \quad\backslash\quad\right]$$

we can generalize the modularity to community splits that have c communities with the following equation

$$Q = \frac{1}{2m}\sum_{i=0}^{n-1}\sum_{j=0}^{n-1}\left[A_{ij} - \frac{k_i k_j}{2m}\right]\delta(c_i, c_j)$$

where $\delta(c_i, c_j) = \begin{cases} 1 & \text{if } c_i = c_j \end{cases}$

$\delta(c_i, c_j) = \{ \; 0 \quad \text{if} \quad c_i \neq c_j$

Let's look at one community detection algorithm: the Girvan-Newman algorithm. It is an algorithm that detects algorithms by progressively removing edges from the network.

To describe this algorithm we first have to describe edge betweenness (a centrality measure for edges in a network). It's defined as the sum of the number of shortest paths passing through the edge in question. For an edge e, the betweeness centrality $C_B(e)$ is:

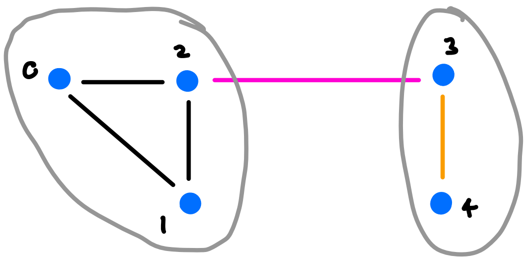
$$C_B(e) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \frac{\sigma(i,j \mid e)}{\sigma(i,j)}$$

where $\sigma(i,j)$ is the # of shortest paths from $i \to j$
$\sigma(i,j \mid e)$ is the # of shortest paths from $i \to j$ that pass through edge e.

Girvan-Newman algorithm steps:

1. Calculate the edge betweeness for all edges
2. Remove the edge(s) with the largest betweeness
3. Recalculate the edge betweeness
4. Repeat steps 2 and 3 until some condition is met
   - ex) modularity reaches a specific value
   - ex) visually, when the communities are meaningful

Main idea: edges with a large betweeness seperate communities

ex)


First calculate edge betweeness for pink edge
Second calculate edge betweenness for orange edge
To do this we need to calculate all of the shortest paths between nodes

shortest path

$0 \to 1$:	$0 \to 1$			$3 \to 0$:	$3 \to 2 \to 0$	*	
$0 \to 2$:	$0 \to 2$			$3 \to 1$:	$3 \to 2 \to 1$	*	
$0 \to 3$:	$0 \to 2 \to 3$	*		$3 \to 2$:	$3 \to 2$	*	
$0 \to 4$:	$0 \to 2 \to 3 \to 4$	*	*	$3 \to 4$:	$3 \to 4$		*
$1 \to 0$:	$1 \to 0$			$4 \to 0$:	$4 \to 3 \to 2 \to 0$	*	*
$1 \to 2$:	$1 \to 2$			$4 \to 1$:	$4 \to 3 \to 2 \to 1$	*	*
$1 \to 3$:	$1 \to 2 \to 3$	*		$4 \to 2$:	$4 \to 3 \to 2$	*	*
$1 \to 4$:	$1 \to 2 \to 3 \to 4$	*	*	$4 \to 3$:	$4 \to 3$		*

2→0:	2→0	
2→1:	2→1	
2→3:	2→3	*
2→4:	2→3→4	* *

find the # of the edges that pass through the pink edge (denoted by a *)
find the # of edges that pass through the orange edge (denoted by a *)

of * : 12
of * : 8

If we do the same thing for all edges we would find that the pink edge has the highest edge betweeness so this algorithm would get rid of the pink edge.

Note: for a given edge, if the edge betweenness is large, that means its probably separating two communities

There are lots of other (more-complicated) community detection algorithms:
- Louvain community detection - optimizes modularity
- surprise community detection - optimizes surprise
- Walktrap community detection - capitalizes on the fact that random walks on a network stay within communities
- Heirarchical Agglomerative clustering - iteratively group nodes based on their similarity

  ↳ This is a "clustering" method which we might briefly cover later on.

Project suggestions:
- test community detection algorithms on a network
- develop your own community detection algorithm

↓ See a project from last year!

Identifying Political Clusters Through Network Analysis

James Courtenay, Alexander Ellis, Tristan Scharfenstein under supervision of Ilya Amburg at Cornell University

Our Topic

Democrat and Republican voters alike tend to have strong allegiances to their respective parties. But how strong are those allegiances for the legislators themselves? Our research involved looking at a vast array of political data and using algorithms to see if the resulting clusters would resemble how our partisan political system has grouped our politicians.

Network Analysis & Graph Theory

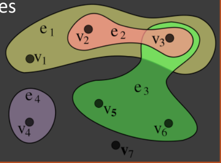
Graph - A collection of nodes, connected to one another by edges

Node - An individual datapoint

Hyperedge - A connection between two or more nodes

Adjacency Matrix - A matrix that represents the nodes' relationships to each other.

Cluster - A grouping of nodes within a graph.

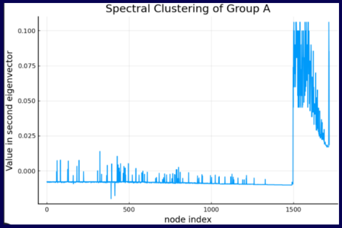


Spectral Clustering

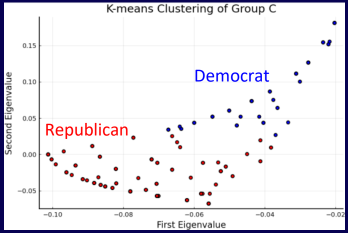
Spectral Clustering uses the concepts of eigenvalues and eigenvectors to approximate clusters for a set of data. If you were to multiply your adjacency matrix A by a matrix x, and received Ax = ƛx, where ƛ is a constant, x is considered to be the eigenvector of A, with the constant ƛ acting as your eigenvalue. For our research, we looked at the second non-zero eigenvalue, or the Fiedler value of a modified adjacency matrix to gain some insight into the connectivity of our graph.

K-means Clustering

K-means is one type of clustering method that simply puts a group of data points together to find patterns from the data. For our project we used k-means clustering to separate different senators and house representatives based on the bills they sponsored. Then from these groups, we determined which groups consisted of a majority of individuals from each political party.



Through both clustering methods, we were able to determine the connectivity and major clusters of our networks. We received a Fiedler value of 763, indicating that we would have to remove 763 nodes from our graph to obtain two completely isolated clusters. Additionally, through analysis of multiple different categorical groupings of our data, we were able to separate most of them into clusters that mainly consisted of either party. These conclusions are evidence of the ever-changing political landscape and show us that politics will not always be black and white, but rather is an extremely dynamic study area affected by many variables. Future research on this topic can aim to find more clusters or categorize data based on other factors.


K-means Clustering of Group C
Democrat
Republican
Second Eigenvalue
First Eigenvalue