**SOCIAL NETWORK ANALYSIS LAB ASSESSMENT-2 SOCIAL**

|  |
| --- |
| **Name: Matam Tejaswini**  **Regno: 22MCB0034** |

**Aim: Implement community detection system in social networks in python.**

Process for implementing community detection in social networks, along with some theory and insights about community detection and the importance of communities in social networks:

**1. Network Representation:**

Start by representing the social network as a graph, where nodes represent individuals or entities, and edges represent relationships or interactions between them. Each node can have attributes such as demographics or interests associated with it.

**2. Community Detection Algorithms:**

There are several popular community detection algorithms that you can use to analyze social networks. Here are a few commonly used ones:

1. Louvain Method: The Louvain method is a fast and widely used algorithm for community detection. It maximizes modularity, which measures the strength of division of a network into communities. The Louvain method iteratively optimizes modularity by moving nodes between communities. You can use the `python-louvain` library to implement the Louvain method in Python.

2. Girvan-Newman Algorithm: The Girvan-Newman algorithm is a hierarchical divisive algorithm that iteratively removes edges with the highest betweenness centrality to detect communities. It repeatedly removes edges until the network breaks into individual communities. The betweenness centrality of an edge measures the number of shortest paths between pairs of nodes that pass through that edge. NetworkX provides an implementation of this algorithm.

3. Infomap: Infomap is an algorithm based on the concept of information theory. It uses a bottom-up approach to identify communities by optimizing a compression algorithm's description length. Infomap maximizes the information flow within communities while minimizing the flow between them. The `python-igraph` library provides an implementation of the Infomap algorithm.

4. Label Propagation Algorithm (LPA): The LPA is a simple and fast algorithm that propagates labels in the network based on the majority of neighbors' labels. Initially, each node is assigned a unique label, and in each iteration, nodes update their labels based on the labels of their neighbors. The process continues until a stable labeling is reached. NetworkX provides a label propagation implementation.

5. Spectral Clustering: Spectral clustering is a technique that uses the eigenvectors of the graph Laplacian matrix to partition the network into communities. It involves creating a similarity matrix based on the network structure, computing the graph Laplacian matrix, and performing spectral decomposition to extract the eigenvectors. The resulting eigenvectors are then used for clustering. The `scikit-learn` library provides an implementation of spectral clustering.

These are just a few examples of community detection algorithms. The choice of algorithm depends on the characteristics of our network and the specific goals of our analysis.

**Implementation of Spectral Clustering:**

To implement a community detection system in social networks using Python, you can use the NetworkX library, which provides useful tools for working with graphs. Here's a step-by-step guide on how to do it:

Install NetworkX: You can install NetworkX using pip by running the following command in your terminal or command prompt:

import numpy as np

from sklearn.cluster import SpectralClustering

import networkx as nx

# Create a graph using NetworkX

G = nx.Graph()

# Add edges to the graph (you can replace this with your own data)

G.add\_edges\_from([(1, 2), (1, 3), (2, 3), (3, 4), (4, 5), (4, 6)])

# Create the adjacency matrix

adjacency\_matrix = nx.adjacency\_matrix(G)

# Convert the adjacency matrix to a numpy array

adjacency\_array = adjacency\_matrix.toarray()

# Perform spectral clustering

n\_clusters = 2  # Set the number of clusters

spectral = SpectralClustering(n\_clusters=n\_clusters, affinity='precomputed')

labels = spectral.fit\_predict(adjacency\_array)

# Print the cluster assignment for each node

for node, label in enumerate(labels):

    print(f"Node {node} belongs to cluster {label}")

# Visualize the communities (optional, requires matplotlib)

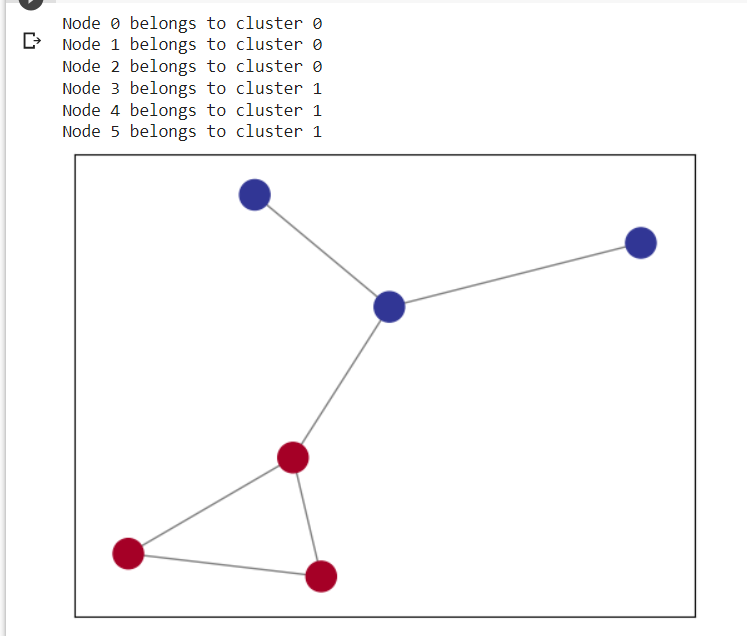
import matplotlib.pyplot as plt

pos = nx.spring\_layout(G)

nx.draw\_networkx\_nodes(G, pos, node\_color=labels, cmap=plt.cm.RdYlBu)

nx.draw\_networkx\_edges(G, pos, alpha=0.5)

plt.show()



**Implementing Community using Girvan-Newman algorithm**

import networkx as nx

from networkx.algorithms import community

# Create a graph using NetworkX

G = nx.Graph()

# Add edges to the graph (you can replace this with your own data)

G.add\_edges\_from([(1, 2), (1, 3), (2, 3), (3, 4), (4, 5), (4, 6)])

# Perform Girvan-Newman algorithm

def girvan\_newman(graph):

    # Calculate the betweenness centrality of each edge

    edge\_betweenness = nx.edge\_betweenness\_centrality(graph)

    # Sort edges by their betweenness centrality in descending order

    sorted\_edges = sorted(edge\_betweenness.items(), key=lambda x: x[1], reverse=True)

    # Iterate over the edges with highest betweenness centrality and remove them

    for edge, \_ in sorted\_edges:

        graph.remove\_edge(\*edge)

        yield nx.connected\_components(graph)

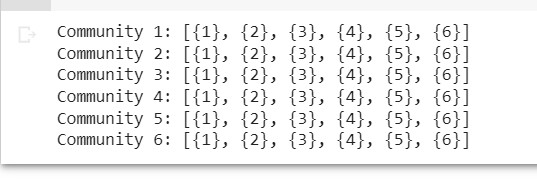
# Get the communities using Girvan-Newman algorithm

communities = tuple(girvan\_newman(G))

# Print the communities

for i, community in enumerate(communities, start=1):

    print(f"Community {i}: {list(community)}")



**Implementing Community using Louvain algorithm**

import networkx as nx

!pip install python-louvain

import community.community\_louvain as cl

# Create a graph using NetworkX

G = nx.Graph()

# Add edges to the graph (you can replace this with your own data)

G.add\_edges\_from([(1, 3), (1, 4), (1, 2), (2, 4), (4, 6), (5, 6), (5, 7)])

# Detect communities using the Louvain method

partition = cl.best\_partition(G)

# Print the community membership of each node

for node, community\_id in partition.items():

    print(f"Node {node} belongs to community {community\_id}")

# Visualize the communities (optional, requires matplotlib)

import matplotlib.pyplot as plt

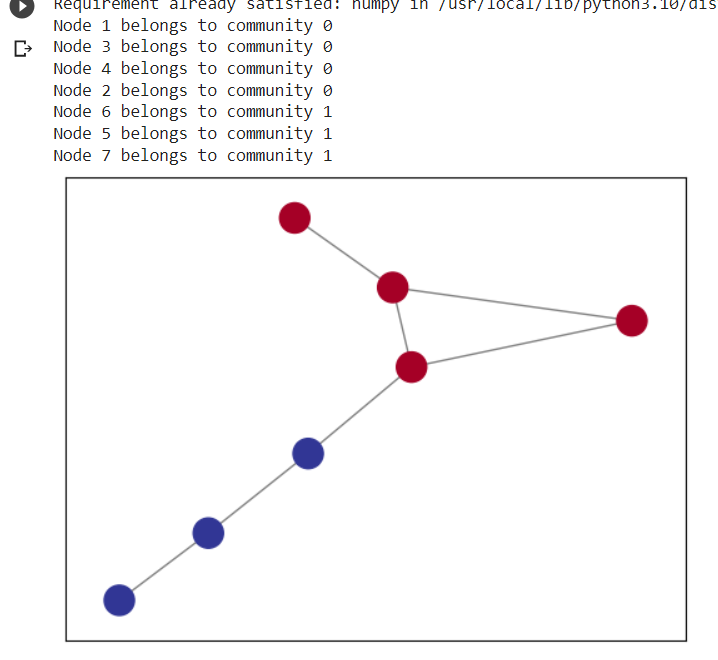
pos = nx.spring\_layout(G)

colors = [partition[n] for n in G.nodes()]

nx.draw\_networkx\_nodes(G, pos, node\_color=colors, cmap=plt.cm.RdYlBu)

nx.draw\_networkx\_edges(G, pos, alpha=0.5)

plt.show()

****