**Social Network analysis**

Digital Assessment

**Name**: Aditya Sharma A

**RegNo:** 22MCB0014

Community detection algorithm

**Newman-Girvan algorithm**

The Newman-Girvan technique, also known as the Girvan-Newman algorithm or the edge betweenness algorithm, is a prominent way for finding communities in networks. Mark Newman and Michelle Girvan founded it in 2002.

The method is based on the concept of "edge betweenness centrality," which measures the importance of edges in connecting diverse communities.

The key idea is that edges with a high betweenness centrality are more likely to occur between communities and hence might be considered prospective community boundaries.

The steps of the Newman-Girvan algorithm are as follows:

Calculate the betweenness centrality for each network edge. This involves calculating how many of the shortest paths between all pairs of nodes pass via each edge.

Remove the edge with the highest betweenness centrality from the network. This effectively splits the network in half or more.

Calculate the centrality of betweenness for each remaining edge in the redesigned network.

Steps 2 and 3 should be repeated until all edges have been removed or the desired number of communities have been identified.

Communities or subnetworks are the resultant disconnected components.

import os

import networkx as nx

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.decomposition import LatentDirichletAllocation

import matplotlib.pyplot as plt

# Step 1: Load the text dataset

with open('/content/group.csv', 'r') as file:

    documents = file.readlines()

# Step 2: Preprocess the text data

# Step 3: Create a document-term matrix

vectorizer = CountVectorizer()

X = vectorizer.fit\_transform(documents)

# Step 4: Apply LDA for topic modeling

lda = LatentDirichletAllocation(n\_components=5)  # Assuming 5 topics

lda.fit(X)

# Step 5: Extract topic distributions for documents

topic\_dist = lda.transform(X)

topic\_labels = topic\_dist.argmax(axis=1)

# Step 6: Create a graph representation of the documents

G = nx.Graph()

for i, document in enumerate(documents):

    G.add\_node(i, text=document, topic=topic\_labels[i])

# Step 7: Apply the Girvan-Newman algorithm for community detection

communities = nx.community.girvan\_newman(G)

# Step 8: Get the final community partition

partition = next(communities)

# Step 9: Visualize the graph with community colors

pos = nx.spring\_layout(G)

# Draw nodes with different community colors

node\_colors = [idx for idx, comm in enumerate(partition) for \_ in comm]

nx.draw\_networkx\_nodes(G, pos, node\_color=node\_colors, cmap='viridis', node\_size=100)

# Draw edges

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

# Draw node labels

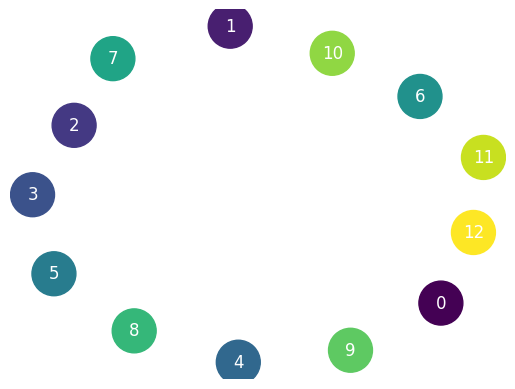
nx.draw\_networkx\_labels(G, pos, font\_color='white')

# Show the plot

plt.axis('off')

plt.show()

OUTPUT:



1. Program code:

import networkx as nx

import matplotlib.pyplot as plt

from networkx.algorithms import community

# Generate a random graph

G = nx.erdos\_renyi\_graph(50, 0.1)

# Apply the Newman-Girvan algorithm

comp = community.girvan\_newman(G)

# Select the top-level community

top\_level\_communities = next(comp)

# Convert the top-level community into a dictionary

partition = {}

for idx, community\_nodes in enumerate(top\_level\_communities):

    for node in community\_nodes:

        partition[node] = idx

# Create a layout for visualizing the graph

layout = nx.spring\_layout(G)

# Draw the nodes with community colors

node\_colors = [partition[node] for node in G.nodes()]

nx.draw\_networkx\_nodes(G, layout, node\_color=node\_colors, cmap='viridis')

# Draw the edges

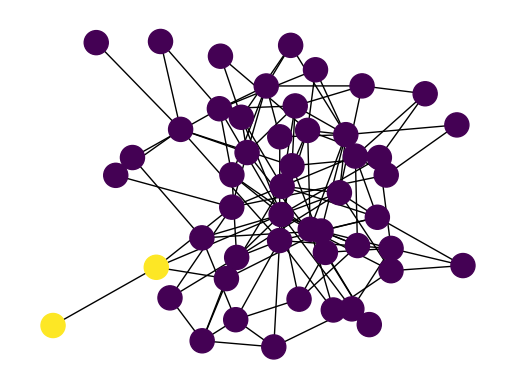
nx.draw\_networkx\_edges(G, layout)

# Display the graph

plt.axis("off")

plt.show()

OUTPUT:



**Louvain algorithm**

The Louvain algorithm is a popular community detection algorithm that aims to find the modular structure of a network. It is an iterative algorithm that optimizes the modularity of the network by iteratively merging and reassigning nodes to communities. Here are the general steps of the Louvain algorithm:

Initialization: Start with each node in the network assigned to its own community.

Modularity calculation: Calculate the modularity of the network in its current state. Modularity measures the quality of a division of a network into communities. It compares the number of edges within communities to the expected number of edges in a random network.

Iteration:

For each node, calculate the modularity gain of removing it from its current community and placing it in the neighboring communities. The modularity gain is the improvement in modularity that would result from moving the node.

Move the node to the community that gives the maximum modularity gain. If the modularity gain is negative or zero for all neighboring communities, leave the node in its current community.

Repeat the above steps for all nodes in the network.

Community aggregation: After completing one pass over all nodes, each community is considered as a single node in a new network. The weights of the edges between communities are the sum of the weights of the original edges between the nodes in the communities.

Repeat steps 2 to 4: Iterate the process of modularity calculation, node reassignment, and community aggregation until no further improvement in modularity is observed. This means that the algorithm has converged and found the optimal community structure.

Post-processing: Once the Louvain algorithm has converged, the final community structure is obtained. Nodes within the same community are grouped together.

It's worth noting that the Louvain algorithm is a heuristic method and may not always find the global optimum, but it often produces good results in practice

Program Code:

import os

import networkx as nx

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.decomposition import LatentDirichletAllocation

from community import community\_louvain

import matplotlib.pyplot as plt

# Step 1: Load the text dataset

with open('/content/group.csv', 'r') as file:

    documents = file.readlines()

# Step 2: Preprocess the text data

# Step 3: Create a document-term matrix

vectorizer = CountVectorizer()

X = vectorizer.fit\_transform(documents)

# Step 4: Apply LDA for topic modeling

lda = LatentDirichletAllocation(n\_components=5)  # Assuming 5 topics

lda.fit(X)

# Step 5: Extract topic distributions for documents

topic\_dist = lda.transform(X)

topic\_labels = topic\_dist.argmax(axis=1)

# Step 6: Create a graph representation of the documents

G = nx.Graph()

for i, document in enumerate(documents):

    G.add\_node(i, text=document, topic=topic\_labels[i])

# Step 7: Apply the Louvain algorithm for community detection

partition = community\_louvain.best\_partition(G)

# Step 8: Visualize the graph with community colors

pos = nx.spring\_layout(G)

# Get unique community labels

community\_labels = set(partition.values())

# Draw nodes with different community colors

node\_colors = [partition[node] for node in G.nodes()]

nx.draw\_networkx\_nodes(G, pos, node\_color=node\_colors, cmap='plasma', node\_size=500)

# Draw edges with community colors

edge\_colors = ['blue' if partition[edge[0]] != partition[edge[1]] else 'viridis' for edge in G.edges()]

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

# Draw node labels

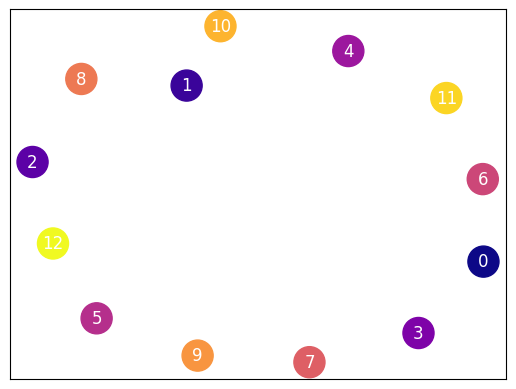
nx.draw\_networkx\_labels(G, pos, font\_color='white')

# Show the plot

plt.axis('on')

plt.show()

OUTPUT:



1. Program code:

from community import community\_louvain

import matplotlib.cm as cm

import matplotlib.pyplot as plt

import networkx as nx

# Generate a random graph

G = nx.erdos\_renyi\_graph(50, 0.1)

# Apply the Louvain community detection algorithm

partition = community\_louvain.best\_partition(G)

# Create a layout for visualizing the graph

layout = nx.spring\_layout(G)

# Draw the nodes with community colors

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

    nx.draw\_networkx\_nodes(G, layout, [node], node\_color=f"C{community\_id+1}")

# Draw the edges

nx.draw\_networkx\_edges(G, layout)

# Display the graph

plt.axis("off")

plt.show()

OUTPUT:

