



VIT[®]
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

**School of Computer Science and Engineering
(WINTER 2022-2023)**

Student Name: **RUDRAJIT DAS**

Reg No: **22MCB0008**

Email: rudrajit.das2022@vitstudent.ac.in
Mobile: 9831060863

Subject Name: **SOCIAL NETWORK ANALYTICS LAB**

Assessment No: 2

Community detection algorithm

1. Newman-Girvan algorithm

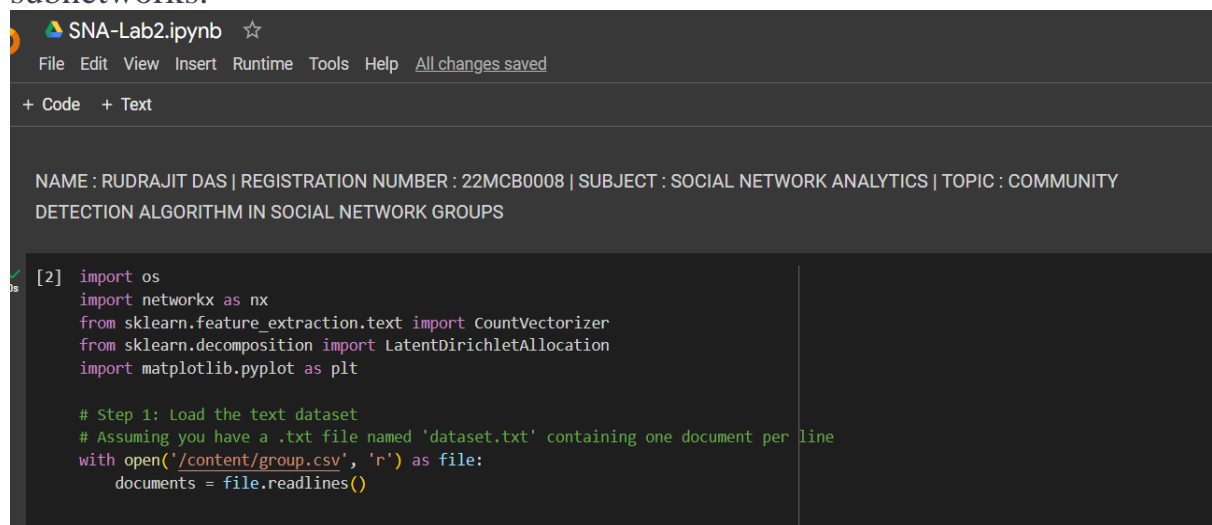
The Newman-Girvan algorithm, also known as the Girvan-Newman algorithm or the edge betweenness algorithm, is a widely used algorithm for community detection in networks. It was developed by Mark Newman and Michelle Girvan in 2002.

The algorithm is based on the concept of "edge betweenness centrality," which measures the importance of edges in connecting different communities. The basic idea is that edges with high betweenness centrality are more likely to lie between communities and therefore can be considered as potential boundaries between communities.

Here are the steps of the Newman-Girvan algorithm:

1. Calculate the betweenness centrality for all edges in the network. This involves finding the shortest paths between all pairs of nodes and determining how many of those paths pass through each edge.
2. Remove the edge with the highest betweenness centrality from the network. This effectively disconnects the network into two or more components.
3. Calculate the betweenness centrality for all remaining edges in the updated network.
4. Repeat steps 2 and 3 until all edges have been removed or until a desired number of communities have been identified.

The resulting disconnected components are considered as communities or subnetworks.



The screenshot shows a Jupyter Notebook titled "SNA-Lab2.ipynb". The notebook has a menu bar with "File", "Edit", "View", "Insert", "Runtime", "Tools", "Help", and "All changes saved". Below the menu bar, there are tabs for "+ Code" and "+ Text". The main content area displays the following code:

```
[2] 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
# Assuming you have a .txt file named 'dataset.txt' containing one document per line
with open('/content/group.csv', 'r') as file:
    documents = file.readlines()
```

```
[3] # Step 2: Preprocess the text data
     # Assuming you have already preprocessed the text data by removing stopwords, punctuation, etc.
     # If not, you can implement preprocessing steps here
```

```
[4] # Step 3: Create a document-term matrix
     vectorizer = CountVectorizer()
     X = vectorizer.fit_transform(documents)
```

```
[5] # Step 4: Apply LDA for topic modeling
     lda = LatentDirichletAllocation(n_components=5) # Assuming 5 topics
     lda.fit(X)
```

▼ LatentDirichletAllocation
LatentDirichletAllocation(n_components=5)

```
✓ [6] # Step 5: Extract topic distributions for documents
0s topic_dist = lda.transform(X)
    topic_labels = topic_dist.argmax(axis=1)
```

```
✓ [7] # Step 6: Create a graph representation of the documents
0s G = nx.Graph()
    for i, document in enumerate(documents):
        G.add_node(i, text=document, topic=topic_labels[i])
```

```
✓ [8] # Step 7: Apply the Girvan-Newman algorithm for community detection
0s communities = nx.community.girvan_newman(G)
```

```
✓ [9] # Step 8: Get the final community partition
0s partition = next(communities)
```

```
✓ [10] # Step 9: Visualize the graph with community colors
0s pos = nx.spring_layout(G)
```

```
✓ [11] # 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)

# 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

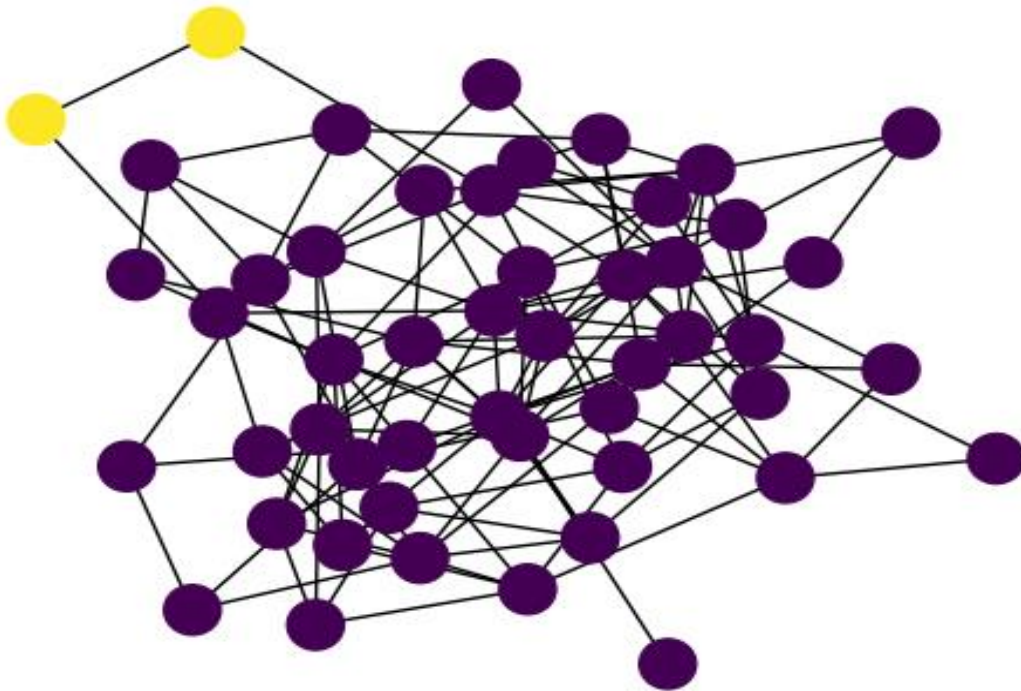
# Create a layout for visualizing the graph
layout = nx.spring_layout(G)

# 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:



1. **import networkx as nx**: This imports the NetworkX library, which provides data structures and algorithms for graph analysis.
2. **import matplotlib.pyplot as plt**: This imports the pyplot module from the Matplotlib library, which is used for graph visualization.
3. **from networkx.algorithms import community**: This imports the community module from NetworkX, which contains algorithms for community detection.
4. **G = nx.erdos_renyi_graph(50, 0.1)**: This line generates a random graph using the Erdos-Renyi model. It creates a graph with 50 nodes and an edge probability of 0.1.
5. **comp = community.girvan_newman(G)**: This line applies the Newman-Girvan algorithm (Girvan-Newman algorithm) to the graph **G** using the **girvan_newman** function from the community module. It returns an iterator over sets of communities at each level of the algorithm.
6. **top_level_communities = next(comp)**: This line extracts the top-level communities from the iterator obtained in the previous step using the **next** function. It represents the community structure at the highest level of division.
7. **partition = {}**: This creates an empty dictionary to store the community assignments for each node.

8. **for idx, community_nodes in enumerate(top_level_communities):**
This loop iterates over each community in the top-level communities and assigns a unique index to each community.
9. **for node in community_nodes: partition[node] = idx:** This nested loop assigns the community index (**idx**) to each node in the respective community.
10. **layout = nx.spring_layout(G):** This line creates a layout for the graph visualization using the spring layout algorithm provided by NetworkX. It assigns positions to the nodes in a way that minimizes the total potential energy of the edges.
11. **node_colors = [partition[node] for node in G.nodes()]:** This line creates a list of community indices corresponding to each node in the graph.
12. **nx.draw_networkx_nodes(G, layout, node_color=node_colors, cmap='viridis'):** This line draws the nodes of the graph using NetworkX's **draw_networkx_nodes** function. It assigns the community colors to the nodes based on the **node_colors** list. The **cmap='viridis'** argument specifies the colormap to use.
13. **nx.draw_networkx_edges(G, layout):** This line draws the edges of the graph using NetworkX's **draw_networkx_edges** function.
14. **plt.axis('off'):** This line turns off the axes for the plot.
15. **plt.show():** This line displays the graph plot.

The code generates a random graph, applies the Newman-Girvan algorithm to detect communities, assigns community colors to the nodes, and visualizes the graph with nodes colored based on their communities.

2. 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)

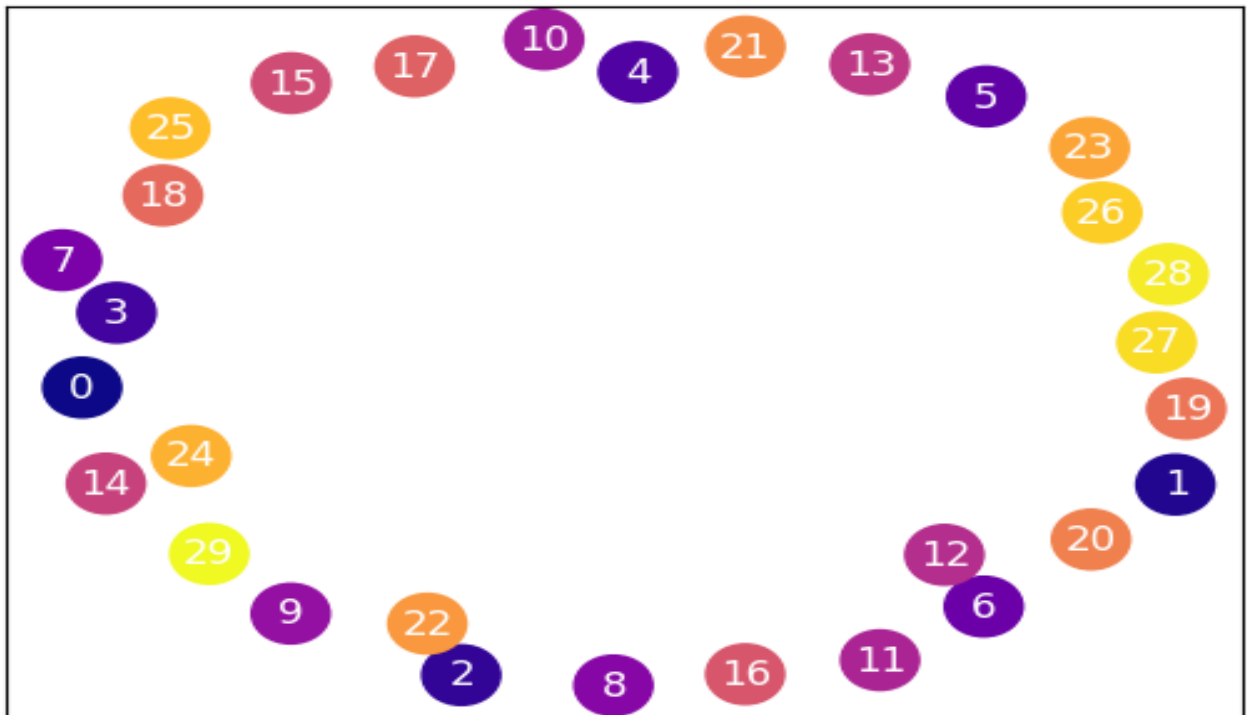
# Draw node labels
nx.draw_networkx_labels(G, pos, font_color='white')

# Show the plot
plt.axis('on')
plt.show()
```


This code performs topic modeling on a text dataset using Latent Dirichlet Allocation (LDA) and visualizes the resulting graph with community colors using the Louvain algorithm. Here's an explanation of each step:

1. **Load the text dataset:** The code reads the contents of the 'group.csv' file into the `documents` list.
2. **Preprocess the text data:** This step involves any necessary preprocessing of the text data, such as removing stop words, stemming, or cleaning the text. It is missing from the provided code and should be implemented depending on the specific requirements of the text dataset.
3. **Create a document-term matrix:** The code uses the `CountVectorizer` from scikit-learn to convert the preprocessed text data into a document-term matrix `X`. Each row of `X` represents a document, and each column represents a unique term in the dataset.
4. **Apply LDA for topic modeling:** The code initializes an LDA model with 5 topics (adjustable based on your specific needs) and fits it to the document-term matrix `X` using the `fit` method.
5. **Extract topic distributions for documents:** The code applies the trained LDA model to the document-term matrix `X` using the `transform` method. It obtains the topic distribution for each document and determines the most probable topic for each document based on the highest probability.
6. **Create a graph representation of the documents:** The code creates an empty graph `G` using NetworkX. It iterates over the documents and adds a node to the graph for each document. Each node is assigned the document's text and the corresponding topic label.
7. **Apply the Louvain algorithm for community detection:** The code applies the Louvain algorithm from the `community` module to detect communities in the graph `G`. It assigns a community label to each node in the graph based on the detected communities.
8. **Visualize the graph with community colors:** The code prepares the layout of the graph using the spring layout algorithm (`pos = nx.spring_layout(G)`). It then visualizes the graph by drawing nodes with different community colors using the `draw_networkx_nodes` function. The node colors are determined based on the community labels obtained from the Louvain algorithm. The code also draws edges with different colors based on whether the connected nodes belong to the same community or not. Node labels are added to the graph using the `draw_networkx_labels` function. Finally, the plot is displayed using `plt.show()`.

OUTPUT:



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()
```

The provided code applies the Louvain community detection algorithm to a randomly generated graph and visualizes the graph with nodes colored according to their communities. Here's an explanation of each step:

1. **from community import community_louvain**: This line imports the **community_louvain** module, which contains the implementation of the Louvain community detection algorithm.
2. **import matplotlib.cm as cm**: This line imports the **cm** module from Matplotlib, which provides colormaps for visualizations.
3. **import matplotlib.pyplot as plt**: This line imports the **pyplot** module from Matplotlib, which is used for graph visualization.
4. **import networkx as nx**: This line imports the NetworkX library, which provides data structures and algorithms for graph analysis.
5. **G = nx.erdos_renyi_graph(50, 0.1)**: This line generates a random graph using the Erdos-Renyi model. It creates a graph with 50 nodes and an edge probability of 0.1.
6. **partition = community_louvain.best_partition(G)**: This line applies the Louvain community detection algorithm to the graph **G** using the **best_partition** function from the **community_louvain** module. It returns a dictionary where the keys are nodes and the values are the corresponding community IDs.
7. **layout = nx.spring_layout(G)**: This line creates a layout for the graph visualization using the spring layout algorithm provided by NetworkX. It assigns positions to the nodes in a way that minimizes the total potential energy of the edges.
8. **for node, community_id in partition.items():**: This loop iterates over each node in the graph and its corresponding community ID.
9. **nx.draw_networkx_nodes(G, layout, [node], node_color=f"C{community_id+1}')**: This line draws the node with its community color using the **draw_networkx_nodes** function from NetworkX. The **node_color** parameter is set to a string representing the community color based on the community ID.
10. **nx.draw_networkx_edges(G, layout)**: This line draws the edges of the graph using NetworkX's **draw_networkx_edges** function.
11. **plt.axis('off')**: This line turns off the axes for the plot.
12. **plt.show()**: This line displays the graph plot.

The code applies the Louvain algorithm to detect communities in the random graph and visualizes the graph with nodes colored according to their community assignments.

OUTPUT:

