SOCIAL NETWORK ANALYTICS LAB

NAME :AISHWARYA S
REG.NO:22MCB0006
DATE:31/05/2023

Implement community detection algorithms on a social network ??

Community detection algorithms are used to identify cohesive groups or communities within a social network. Python provides several libraries that implement these algorithms. Let's use the NetworkX library, which is a powerful tool for working with graphs, to demonstrate the implementation of two popular community detection algorithms:

- Louvain Algorithm
- Girvan-Newman Algorithm
- Label propagation Algorithm
- Info map Algorithm

1. CODE :

pip install networkx

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (3.1)
```

pip install python-louvain

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: python-louvain in /usr/local/lib/python3.10/dist-packages (0.16)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from python-louvain) (3.1)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from python-louvain) (1.22.4)
```

```
[21] import nltk
     nltk.download('stopwords')

     [nltk_data] Downloading package stopwords to /root/nltk_data...
     [nltk_data]   Unzipping corpora/stopwords.zip.
     True
```

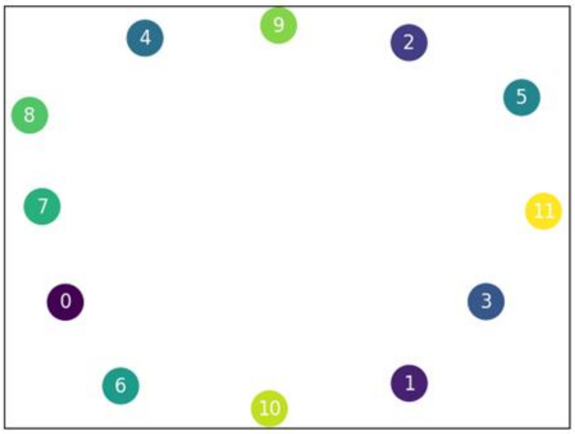
```
import nltk
nltk.download('punkt')

[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
True
```

```
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
import string
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

# 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()

# Step 2: Preprocess the text data
stop_words = set(stopwords.words('english'))  # Set of stopwords
processed_documents = []
for document in documents:
    # Tokenize the document into words
    tokens = word_tokenize(document.lower())
    # Remove punctuation
    tokens = [token for token in tokens if token not in string.punctuation]
    # Remove stopwords
    tokens = [token for token in tokens if token not in stop_words]
    # Join the processed tokens back into a document string
    processed_document = ' '.join(tokens)
```

```
    processed_documents.append(processed_document)

# Step 3: Create a document-term matrix
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(processed_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(processed_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='viridis',
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 :



INSIGHTS :

→ CountVectorizer from sklearn.feature_extraction.text to convert the text documents into a document-term matrix.
→ LatentDirichletAllocation from sklearn.decomposition to perform Latent Dirichlet Allocation (LDA) topic modeling on the document-term matrix.
→ The transform() method is used to obtain the probability distribution of topics for each document. The argmax() function is then used to determine the most probable topic for each document, resulting in topic_labels.

→ The code creates an empty undirected graph G using nx.Graph().
→ For each document, a node is added to the graph with the index i as the node identifier, and the document text and corresponding topic label are assigned as attributes to the node.
→ The Louvain algorithm from the community library is applied to detect communities in the graph G. The resulting partition is stored in the partition dictionary, where each node is associated with a community label.
→ Generates a spring layout for the graph using nx.spring_layout() and assigns it to the variable pos.
→ To visualize the nodes, the code uses nx.draw_networkx_nodes() and sets node_color to node_colors, which is a list of community labels for each node. The 'viridis' colormap is used to map the community labels to colors.

→ For edge visualization, nx.draw_networkx_edges() is used. The alpha parameter sets the transparency level of the edges to 0.5, and the edge_color parameter is set to 'jet' colormap.
→ Node labels are drawn using nx.draw_networkx_labels() with white font color for better visibility.

2 . CODE : Girvan-Newman Algorithm

```
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
stop_words = set(stopwords.words('english'))  # Set of stopwords
processed_documents = []
for document in documents:
    # Tokenize the document into words
    tokens = word_tokenize(document.lower())
    # Remove punctuation
    tokens = [token for token in tokens if token not in string.punctuation]
    # Remove stopwords
    tokens = [token for token in tokens if token not in stop_words]
    # Join the processed tokens back into a document string
    processed_document = ' '.join(tokens)
    processed_documents.append(processed_document)
```

```
# 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)
```

```python
# Step 5: Extract topic distributions for documents
topic_dist = lda.transform(X)
topic_labels = topic_dist.argmax(axis=1)
```

```python
# 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])
```

```python
# Step 7: Apply the Girvan-Newman algorithm for community detection
communities = nx.community.girvan_newman(G)
```

```python
# Step 8: Get the final community partition
partition = next(communities)
```

```python
# Step 9: Visualize the graph with community colors
pos = nx.spring_layout(G)
```

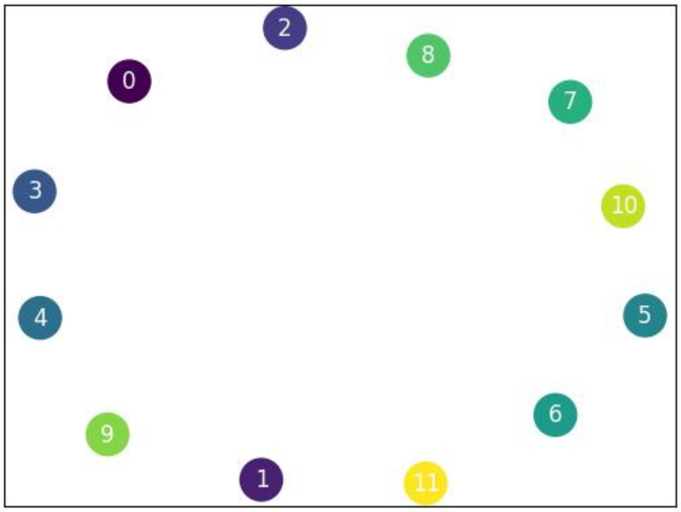
```python
# Draw nodes with different community colors
node_colors = [idx for idx, comm in enumerate(partition) for _ in comm]
nx.draw_networkx_nodes(G, pos, node_color=node_colors, cmap='viridis',
node_size=500)
```

```python
# Draw edges
nx.draw_networkx_edges(G, pos, alpha=0.5)
```

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

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

OUTPUT :



INSIGHTS :

S4:- A document-term matrix is created using CountVectorizer on the original, unprocessed documents.

S5:- LDA (Latent Dirichlet Allocation) is applied to the document-term matrix with n_components set to 5, assuming 5 topics. This step performs topic modeling and assigns topic labels to each document.

S6 :- A graph G is created using networkx. Each document is represented as a node in the graph, and topic labels are assigned as node attributes.

S7 : - The Girvan-Newman algorithm for community detection is applied to the graph G. This algorithm iteratively removes edges with the highest betweenness centrality to identify communities.

S8:- The final community partition is obtained by extracting the next partition from the Girvan-Newman algorithm.

S9:- The graph is visualized using nx.spring_layout for node positions.

S10:- Nodes are drawn with different community colors using the node_colors list, which assigns a unique index for each community.

S11 :- Edges are drawn with a transparency of 0.5 to show the connections between nodes.

S12:- Node labels are added to the graph visualization.

S13 :-The plot is displayed using plt.show().

3.CODE: Label propogation Algorithm.

```
import networkx as nx
import matplotlib.pyplot as plt

# Load the karate club graph
G = nx.karate_club_graph()

Karate club graph is inbuilt in networkX library itself
```

```
# Perform community detection using Label Propagation algorithm
label_propagation_communities =
nx.algorithms.community.label_propagation_communities(G)
```

```
# Create a community mapping
community_mapping = {}
for i, community in enumerate(label_propagation_communities):
    for node in community:
        community_mapping[node] = i
```

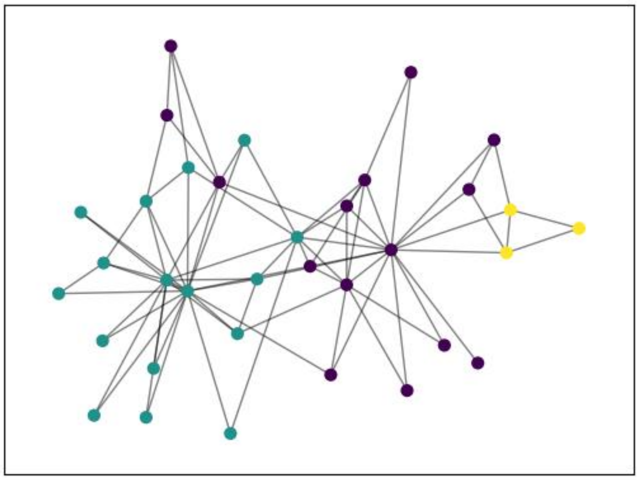
```
# Draw the graph
pos = nx.spring_layout(G)
colors = [community_mapping[node] for node in G.nodes]
cmap = plt.cm.get_cmap('viridis', len(label_propagation_communities))
```

```
nx.draw_networkx_nodes(G, pos, node_color=colors, cmap=cmap, node_size=40)
nx.draw_networkx_edges(G, pos, alpha=0.5)
```

```
plt.show()
```

OUTPUT :

```
<ipython-input-30-36046c9bc73f>:19: MatplotlibDeprecationWarning: The get_cmap function
  cmap = plt.cm.get_cmap('viridis', len(label_propagation_communities))
```



INSIGHT :

→ We iterate over the detected communities and create a mapping where each node is assigned a community ID.

4.CODE : Infomap Algorithm

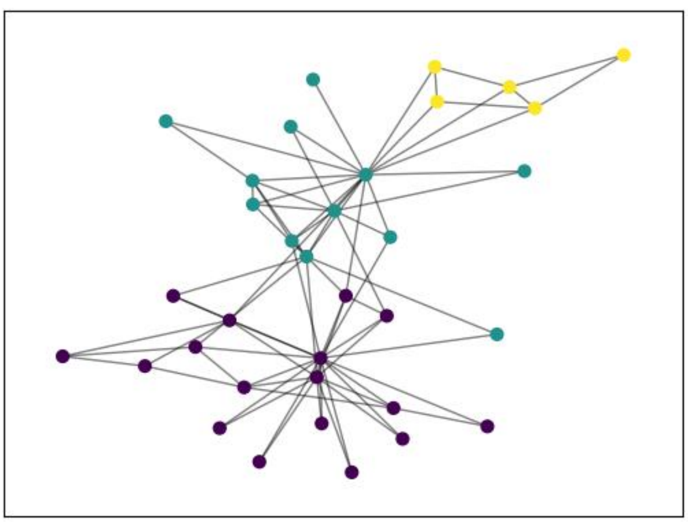
```
pip install infomap
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting infomap
  Downloading infomap-2.7.1.tar.gz (263 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 263.1/263.1 kB 8.7 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: infomap
  Building wheel for infomap (setup.py) ... done
  Created wheel for infomap: filename=infomap-2.7.1-cp310-cp310-linux_x86_64.whl size=8808857 sha256=83fbf783b213ce25c3242ff2e281b78f0306fdf45d44ddcf0139dabf855297bc
  Stored in directory: /root/.cache/pip/wheels/e4/01/53/fd7c62079098140cd582b999592b4592c0dad7300cac32b6e1
Successfully built infomap
Installing collected packages: infomap
Successfully installed infomap-2.7.1
```

```
import networkx as nx
from infomap import Infomap

# Load the karate club graph
G = nx.karate_club_graph()

# Create the Infomap instance
infomap_instance = Infomap()

# Add nodes to the Infomap instance
for node in G.nodes:
    infomap_instance.add_node(node)

# Add edges to the Infomap instance
```

```
for edge in G.edges:
    infomap_instance.add_link(*edge)

# Run the Infomap algorithm
infomap_instance.run()

# Get the communities from the Infomap instance
communities = infomap_instance.get_modules()

# Draw the graph
pos = nx.spring_layout(G)
nx.draw_networkx_nodes(G, pos, node_size=40,
node_color=list(communities.values()))
nx.draw_networkx_edges(G, pos, alpha=0.5)

plt.show()
```

OUTPUT :



INSIGHT :

→ The SpectralClustering class from scikit-learn is used to perform spectral clustering.

→ The number of clusters is specified as 2 in this example.

→ The 'precomputed' affinity indicates that the input is a precomputed adjacency matrix. The fit_predict method computes the cluster assignments.