GT PROJECT

TOPIC: Customer Segmentation Using Graph Clustering

Group Leader:

M.Abdullah Shariq (22K-4497)

Members:

Affan Hassan (22k-4445)

Umer Yousuf (22K-4515)

Project Video Link :

https://www.loom.com/share/360857208876445aaca07c1061222e8e?sid=cb75e489-ad75-4b13-bc81-481d9708795c

Project Details:

1. Data Preprocessing

- **Read Data**: The code reads a dataset (`ecommerce_customer_data_custom_ratios.csv`) containing e-commerce customer data using `pandas`.
- **Sampling**: A random 0.5% subset of the dataset is sampled to make computations manageable.
- **Group Data**: Groups the sampled data by `Customer ID` and `Product Category`, summing up the quantities purchased.

2. Mapping Product Categories

- Converts `Product Category` values into numeric labels (e.g., Books → 1, Electronics → 2) using a predefined mapping. This ensures data consistency for graph processing.

3. Graph Construction

- **Node Creation**:
  - Creates a bipartite graph where:
    - Customers (`Customer ID`) are one set of nodes.
    - Products (`Product Category`) are another set of nodes.
  - Both node types are labeled for clarity.
- **Edge Creation**:
  - Adds edges between customer nodes and product nodes.
  - Edge weights represent the quantity of products purchased by customers.

4. Community Detection

- Uses the **greedy modularity algorithm** to identify communities (clusters) within the graph. These communities represent groups of customers with similar purchasing patterns.

5. Visualization

- Calculates the layout positions for nodes (`spring_layout`).
- Assigns colors to different communities.
- Plots the graph with distinct node colors for each cluster and visualizes the connections (edges).

6. Clustering Results

- **Cluster Output**:
  - Saves the detected clusters as a JSON file (`clusters.json`), where each cluster contains a list of customer IDs or product categories.
- **Modularity**:
  - Computes and prints the modularity score, which measures the strength of the clustering.

7. Graph Metrics

The following metrics are computed and displayed for deeper insights:

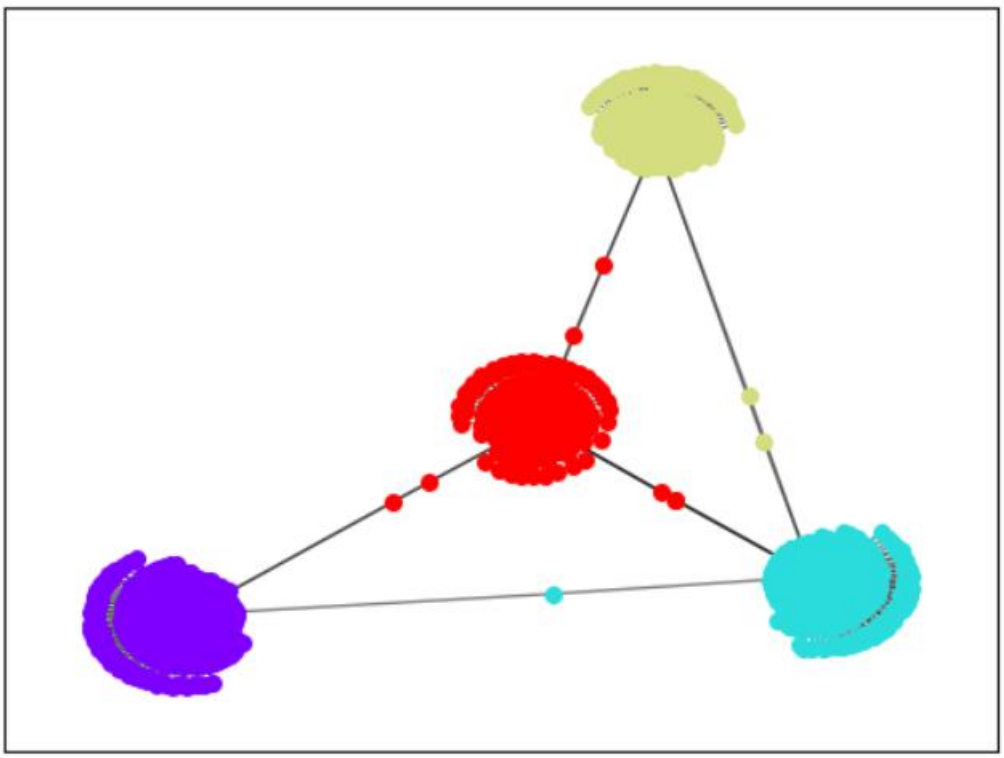
- **Average Degree**: Mean number of connections per node.
- **Diameter**: Longest shortest path between any two nodes in the graph (if the graph is connected).
- **Average Path Length**: Mean shortest path length between all pairs of nodes (if the graph is connected).
- **Clustering Coefficient**: Measures the tendency of nodes to form clusters.

- **Graph Density**: Ratio of existing edges to all possible edges.

Output Highlights

- **Visualization**: Displays a color-coded graph of the clusters.
- **JSON File**: Saves clusters for future analysis.
- **Printed Metrics**:
    - Modularity (effectiveness of clustering).
    - Average degree, diameter, path length, clustering coefficient, and graph density.

OUTPUT:



```
[23]: clusters = {f"Cluster_{i}": list(map(int, community)) for i, community in enum
```

[23]:
```python
clusters = {f"Cluster_{i}": list(map(int, community)) for i, community in enumerate(communities)}
with open("clusters.json", "w") as f:
    json.dump(clusters, f)

modularity = nx.algorithms.community.modularity(G, communities)
print(f"Modularity: {modularity}")
```

```
Modularity: 0.7291464815370003
```

[25]:
```python
avg_degree = np.mean([deg for node, deg in G.degree()])
print(f"Average Degree: {avg_degree}")
```

```
Average Degree: 2.0096618357487923
```


```
Average Degree: 2.0096618357487923
```


[27]:
```python
try:
    diameter = nx.diameter(G)
    print(f"Diameter: {diameter}")
except nx.NetworkXError:
    diameter = "Inf"
    print(f"Diameter: {diameter}")

try:
    avg_path_length = nx.average_shortest_path_length(G)
    print(f"Average Path Length: {avg_path_length}")
except nx.NetworkXError:
    avg_path_length = "Inf"
    print(f"Average Path Length: {avg_path_length}")

clustering_coefficient = nx.average_clustering(G)
print(f"Clustering Coefficient: {clustering_coefficient}")

density = nx.density(G)
print(f"Graph Density: {density}")
```

```
Diameter: 6
Average Path Length: 3.720735835860385
Clustering Coefficient: 0.0
Graph Density: 0.0016193890699023306
```