Community Detection in GitHub User Networks Using AGM

Akshat Raj Saxena
Roll Number: 2022054
GitHub Project Link

1 Introduction

In this project, I analyzes GitHub user networks to identify communities using the AGM algorithm. This project demonstrates the practical application of graph theory and community detection algorithms in social network analysis with a large dataset.

2 Dataset Description

The dataset used in this project consists of GitHub user interactions represented as edges in a graph. The data is stored in a CSV format with the following structure:

```
id_1,id_2
1,2
2,3
...
```

where each row represents an edge between two users identified by their unique IDs.

3 Methodology

3.1 Graph Creation

The first step involves creating a network graph from the edge data. We use NetworkX to construct an undirected graph where nodes represent users and edges represent connections between them. Below is the sample code.

```
import networkx as nx
import csv

edges_list = []
with open("musae_git_edges.csv", 'r') as file:
    csv_reader = csv.DictReader(file)
    for row in csv_reader:
        id_1 = int(row['id_1'])
        id_2 = int(row['id_2'])
        edges_list.append((id_1, id_2))

G = nx.Graph()
G.add_edges_from(edges_list)
```

3.2 Community Detection Algorithm

We implement the Affiliation Graph Model algorithm using NetworkX's built-in functions. The algorithm works by:

1. Initially treating each node as a separate community
2. Iteratively merging communities to maximize modularity
3. Stopping when no further improvement in modularity is possible

```
from networkx.algorithms.community import
    greedy_modularity_communities
communities = list(greedy_modularity_communities(G))
modularity = nx.algorithms.community.quality.modularity(G,
    communities)
```

3.3 Community Assignment

After detecting communities, I assign each node to its respective community and create a mapping for analysis:

```
from collections import defaultdict

communities_dict = defaultdict(list)
for i, community in enumerate(communities):
    for node in community:
        communities_dict[node] = i
from pyspark.sql import SparkSession
from pyspark.sql.types import StructType, StructField, IntegerType

spark =
    SparkSession.builder.appName("CommunityDetectionAGM").getOrCreate()
edges_schema = StructType([
    StructField("id_1", IntegerType(), True),
    StructField("id_2", IntegerType(), True)
])
edges_df = spark.read.csv("musae_git_edges.csv",
                          header=True,
                          schema=edges_schema)
```

4 Results and Analysis

4.1 Community Structure

The algorithm successfully identified distinct communities within the network. Key metrics include:

- Number of communities detected
- Size distribution of communities
- Modularity score of the partition

4.2 Modularity Analysis

The modularity score provides a measure of the quality of the community detection:

$$Q = \frac{1}{2m} \sum_{i,j} [A_{ij} - \frac{k_i k_j}{2m}] \delta(c_i, c_j) \tag{1}$$

where:

- m is the number of edges
- A_{ij} is the adjacency matrix
- k_i is the degree of node i
- c_i is the community of node i
- δ is the Kronecker delta function

5 Working with the Project

To reproduce the results:

1. Clone the repository
2. Install dependencies:

```
pip install networkx numpy pandas pyspark
```

3. Run the community detection script:

```
python community_detection.py
```

6 Conclusion

This project demonstrates the application of the AGM algorithm for community detection in GitHub user networks. The implementation provides both PySpark and Python solutions, making it suitable for various use cases and dataset sizes.