PSG COLLEGE OF TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

19OH01 – SOCIAL AND ECONOMIC NETWORK ANALYSIS

TOPIC: GITHUB RECOMMENDER SYSTEM

TEAM MEMBERS:

18Z335 - NAVEEN P

19Z433 - GOPISANKAR G

19Z461 - RAHUL R

19Z464 - MOULEESWARAN S

Problem Statement:

By using this recommending system, we analyze user similarity and recommend repositories to users using the concept of bipartite graphs. In our project, there are two sets of nodes(Projects or Repositories and Users). By finding similar projects and similarity score between given users, we recommend repositories to users.

Dataset Description:

Our data set is in the form of adjacency list, whereas the first word is the head node and the following nodes are the neighbours of first node. The word begins with 'u' is considered as users in our project and word begin with 'p' is considered as projects or repositories in our project. Total number of nodes in the dataset is 22353 and edges are 20052. In that nodes 10721 are users and 11632 nodes are projects.

Tools Used:

1. API Used:

→ **Google Colaboratory** - Anybody to write and execute arbitrary python code through the browser, and is especially well suited to machine learning, data analysis and education.

2. Library:

→ **Networkx -** NetworkX is a Python library for studying graphs and networks.

Challenges faced:

1. Dataset is in the format of adjacency list. It is hard to modify the dataset to convert for our convenience.

2. We are new to this Google colab API and networkx. It takes time to understand colab environment and some functions in networkx package.

3. Initially, we have faced difficulties in loading the dataset with our concept in Gephi. So, we moved to Google Colab API because of easy implementation.

Contribution of Team Members:

S.NO	Roll.no	Name	Contribution	
1)	18Z335	Naveen P	Dataset Modification,Coding	Report Preparation (Commonly done by using Google Meet)
2)	19Z433	Gopisankar G	Code Part,Documentation	
3)	19Z461	Rahul R	Project Ideas,Documentation	
4)	19Z464	Mouleeswaran S	Graph Visualization,Coding	

Annexure I:

```
→ import networkx as nx
→ G=nx.Graph()
→ G=nx.read_adjlist("dataset.txt")
→ type(G)
→ len(G.nodes())
→ len(G.edges())

→ for n in G.nodes():
    if(n[:1]=='p'):
      G.nodes[n]['bipartite'] = 'projects'
    else:
      G.nodes[n]['bipartite'] = 'users'

→ #To count number of users and projects in our dataset
    def get_nodes_from_partition(G,partition):
        nodes = []
        for n in G.nodes():
            if G.nodes[n]['bipartite'] == partition:
                #if the node belogs to given partition, then the node is added to nodes list
                nodes.append(n)
        return nodes
    print(len(get_nodes_from_partition(G, 'users')))
    print(len(get_nodes_from_partition(G, 'projects')))

→ def shared_partition_nodes(G, n1, n2):
        #Checking whether passed nodes belong to users partition or not
        assert G.nodes[n1]['bipartite'] =='users'
        assert G.nodes[n2]['bipartite'] =='users'

        # Getting neighbors of node1
        nbr1 = G.neighbors(n1)
        # Getting neighors of node2
        nbr2 = G.neighbors(n2)

        # To get shared projects between users
        common = set(nbr1).intersection(nbr2)
        return common

→ def user_similarity(G, u1, u2, proj_nodes):
```

```
        #Checking whether passed nodes belong to users partition or not
        assert G.nodes[u1]['bipartite'] == 'users'
        assert G.nodes[u2]['bipartite'] == 'users'

        # To get shared projects between given users
        shared_nodes = shared_partition_nodes(G, u1, u2)
        # return similarity score
        return len(shared_nodes) / len(proj_nodes)

→ from collections import defaultdict
    def most_similar_users(G, user, user_nodes, proj_nodes):
        #Checking whether passed nodes belong to users partition or not
        assert G.nodes[user]['bipartite'] == 'users'

        # Getting other user nodes
        user_nodes = set(user_nodes)
        user_nodes.remove(user)

        # Creating dictionary
        similarities = defaultdict(list)
        for n in user_nodes:
            similarity = user_similarity(G, user, n, proj_nodes)
            similarities[similarity].append(n)

        # computing maximum similarity
        max_similarity = max(similarities.keys())

        #  returing maximum similarity users
        return similarities[max_similarity]
    user_nodes = get_nodes_from_partition(G, 'users')
    project_nodes = get_nodes_from_partition(G, 'projects')

→ def recommend_repositories(G, from_user, to_user):
    # repositiores of user1
    from_repos = set(G.neighbors(from_user))
    #repositiories of user2
    to_repos = set(G.neighbors(to_user))

    print("Suggesting repositiories to user1")
    print(to_repos.difference(from_repos))
    print("Suggesting repositiories to user2")
    print(from_repos.difference(to_repos))
```

```
→ G1=nx.Graph()

→def draw_graph(user1,user2):
    G1.add_node(user1)
    G1.add_node(user2)
    repos1=G.neighbors(user1)
    repos2=G.neighbors(user2)
    for n in repos1:
      G1.add_node(n)
      G1.add_edge(user1,n)
    for n in repos2:
      G1.add_node(n)
      G1.add_edge(user2,n)
    #nx.draw(G1,with_labels=True)
    nx.draw(G1,node_color="red",with_labels=True,node_size=1000)

→user1='u21'
    user2='u2509'
    print("Number of Shared repositiories")
    print(len(shared_partition_nodes(G,user1,user2)))
    print("Similarity Score")
    project_nodes = get_nodes_from_partition(G, 'projects')
    similarity_score = user_similarity(G,user1,user2, project_nodes)
    print(similarity_score)
    print("Recommending repositories")
    recommend_repositories(G,user1,user2)
    draw_graph(user1,user2)
```

Annexure II:

```
[4]  type(G)

     networkx.classes.graph.Graph

[5]  len(G.nodes())

     22353

[6]  len(G.edges())

     20052
```

```python
#To count number of users and projects in our dataset
def get_nodes_from_partition(G,partition):
    nodes = []
    for n in G.nodes():
        if G.nodes[n]['bipartite'] == partition:
            #if the node belogs to given partition, then the node is added to nodes list
            nodes.append(n)
    return nodes

print(len(get_nodes_from_partition(G, 'users')))
print(len(get_nodes_from_partition(G, 'projects')))
```

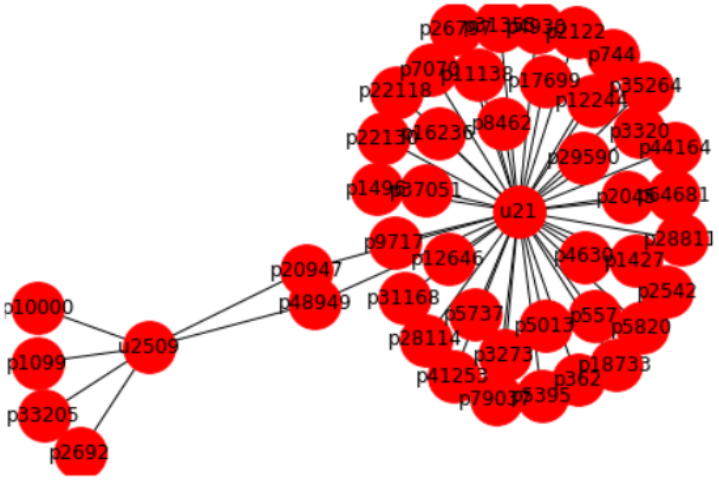
```
10721
11632
```

```
Number of Shared repositiories
2

Similarity Score
0.000171939477303989

Recommending repositories:
Suggesting repositiories to user1
{'p1099', 'p33205', 'p10000', 'p2692'}

Suggesting repositiories to user2
{'p12646', 'p557', 'p8462', 'p12244', 'p3320', 'p35264', 'p28114', 'p26797', 'p362', 'p11138', 'p3273', 'p31355',
```



References:

1. https://towardsdatascience.com/github-recommender-system-python-c8ff64dc83f4

2. https://networkx.org/documentation/stable/tutorial.html

3. https://networkx.org/documentation/latest/_modules/networkx/algorithms/operators/product.html

4. https://campus.datacamp.com/courses/intermediate-network-analysis-in-python/bipartite-graphs-product-recommendation-systems?ex=12

5. https://campus.datacamp.com/courses/intermediate-network-analysis-in-python/bipartite-graphs-product-recommendation-systems?ex=9

6. https://towardsdatascience.com/customizing-networkx-graphs-f80b4e69bedf#:~:text=Altering%20node%20size%20globally%20is,()%20method%20%E2%80%94%20just%20specify%20node_size!