DISCRETE MATHEMATICS

CS101 PROJECT-2

Name : Vivaan Garg
Entry no. : 2023CSB1173

24th April, 2024

Problem Statement

We are tasked with analyzing an impression network provided in the form of a Google Sheet. The objectives are to:

1. Determine the PageRank of each node to identify the top leader of the network.
2. Identify missing links in the network using mathematical formulations and recommend potential links for inclusion.
3. Formulate and explain a new problem based on the dataset.

Question 1

1 PageRank

PageRank is an algorithm developed by Google to measure the importance of webpages based on the quality and quantity of incoming links. It assigns a numerical weight to each element of a hyperlinked set of documents, with the purpose of measuring its relative importance within the set. The algorithm works by treating links as votes, where a page receiving more votes (high-quality links) is considered more significant.

1.1 Random Walking Algorithm

In the random walk method, a node is selected uniformly at random from the graph, and its count is increased by 1. Then, one of its neighbors is chosen uniformly at random, and its count is similarly increased. This process continues iteratively until convergence is reached. Ultimately, the node with the highest count is identified as having the maximum PageRank and is considered the leader of the network or web graph.

1.2 Python Code

Below is the Python code for random walk algorithm:

```python
import random
import csv
import networkx as nx

obj=csv.reader(open('data.csv','r'))
l=[] #nodelist
e=[] #edgelist
next(obj)
G=nx.DiGraph() #directed graph
for row in obj:
    b=row[1][0:11].upper()
    l.append(b)
    for i in row[2:]:
        e.append((b,i[-11:].upper()))
G.add_nodes_from(l) #adding nodes to graph
G.add_edges_from(e) #adding edges to graph

def random_walk(G):
    node=G.nodes()
    coins={}
    for i in node:
        coins[i]=0
    n=random.choice(list(node))
    coins[n]+=1
    x=G.out_edges(n)
    c=0
    while c<300000:
        if len(x)==0:
            m=random.choice(list(node))
        else:
            n1=random.choice(list(x))
            m=n1[1]
        coins[m]+=1
        x=G.out_edges(m)
        c+=1
    return coins
sorted_by_values = sorted(random_walk(G).items(), key=lambda item:
    item[1])
sorted_dict = {key: value for key, value in sorted_by_values}


# function to print top 10 students according to pagerank
def top_ten(sorted_dict):
    f=list(sorted_dict.keys())
    g=list(sorted_dict.values())
    for i in range(-1,-12,-1):
        if f[i]!='':
            print(f[i],g[i])

top_ten(sorted_dict)
```

Listing 1: Random Walking Algorithm

Question 2

2 Missing Links in the Network

2.1 Method

1. **Data Preparation:** We start by loading the network data from a CSV file. The data typically consists of information about nodes and their connections.

2. **Graph Construction:** Using the NetworkX library in Python, we construct a directed graph (DiGraph) representing the network. Nodes and edges are added to the graph based on the information extracted from the CSV file.

3. **Adjacency Matrix Representation:** We create an adjacency matrix to represent the connections between nodes in the graph. Each entry in the adjacency matrix indicates whether there is an edge between the corresponding nodes.

4. **Identifying Missing Links:** To identify missing links, we iterate over each pair of nodes in the graph. If there is no edge between a pair of nodes, we construct a linear system of equations using linear algebra techniques. We remove the row and column corresponding to the current pair of nodes from the adjacency matrix to form a smaller system of equations. Using the least squares method (`np.linalg.lstsq`), we solve the linear system to find the missing link between the nodes. If the resulting value from the linear system is greater than 0, it indicates a potential missing link, and we add the edge to the graph.

5. **Results and Discussion:** We print the identified missing links and count the total number of missing links found. We discuss the implications of identifying missing links in network graphs and how this information can be used for further analysis and prediction.

2.2 Python Code

Below is the Python code for random walk algorithm:

```python
import random
import csv
import networkx as nx
import numpy as np
import scipy as sp

obj=csv.reader(open('data.csv','r'))
l=[] #nodelist
e=[] #edgelist
next(obj)
G=nx.DiGraph() #directed graph
for row in obj:
    b=row[1][0:11].upper()
    l.append(b)
    for i in row[2:]:
        if i=='': continue
        e.append((b,i[-11:].upper()))
G.add_nodes_from(l) #adding nodes to graph
G.add_edges_from(e) #adding edges to graph

l = list(G.nodes())
n = len(l)

adj = []
for i in range(n):
    row = []
    for j in range(n):
        if G.has_edge(l[i], l[j]):
            row.append(1)
        else:
            row.append(0)
    adj.append(row)

adj = np.array(adj)
cnt = 0
for i in range(n):
    for j in range(n):
        if adj[i][j] == 0:
            row = np.delete(adj[i], j, axis=0)
            col = np.delete(adj[:,j], i, axis=0)
            A = np.delete(np.delete(adj, i, axis=0), j, axis=1)
            X = np.linalg.lstsq(A.T, col, rcond=None)[0]
            val = np.matmul(X,col)
            if val > 0:
                G.add_edge(l[i], l[j])
                cnt+=1
                print("Missing Link:", l[i], l[j])
print(cnt)
```

Listing 2: Identifying Missing Links

Question 3

Graph Analysis

Plotting Number of Nodes vs. Number of Indegrees

In this section of the analysis, we examined the relationship between the number of nodes and the number of indegrees in the given network. The purpose was to understand the distribution of indegrees across the nodes and assess whether it follows a particular pattern.

Observation

Upon plotting the graph of the number of nodes versus the number of indegrees, we observed that the resulting plot resembled a normal distribution curve. This means that the majority of nodes in the network tend to have a moderate number of indegrees, with fewer nodes having very high or very low indegree values.

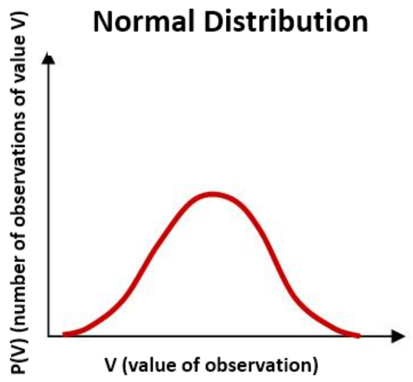
Interpretation

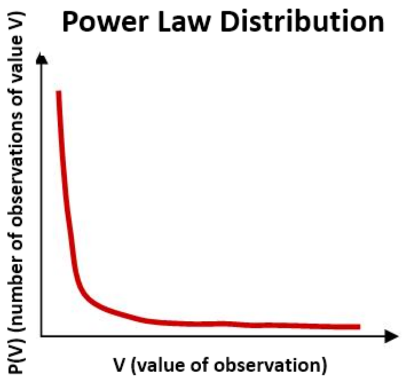
The fact that the plot resembled a normal distribution curve suggests that the distribution of indegrees in the network does not follow a power law. In a power-law distribution, most nodes have very few connections (low indegrees), while a small number of nodes have a disproportionately high number of connections (high indegrees). This characteristic is often observed in scale-free networks.

Implications

The observation that the distribution of indegrees in the network does not follow a power law has several implications:

- **Network Structure:** The network may not exhibit the typical characteristics of a scale-free network, where a few highly connected nodes play a significant role in shaping the network structure.
- **Node Importance:** In a power-law distribution, nodes with high indegrees are considered highly influential or important within the network. However, in this case, the distribution suggests a more balanced influence among nodes, with fewer nodes dominating the network.
- **Connectivity Patterns:** The normal distribution of indegrees implies that most nodes have a similar level of connectivity, which may result in different connectivity patterns compared to networks with power-law distributions.





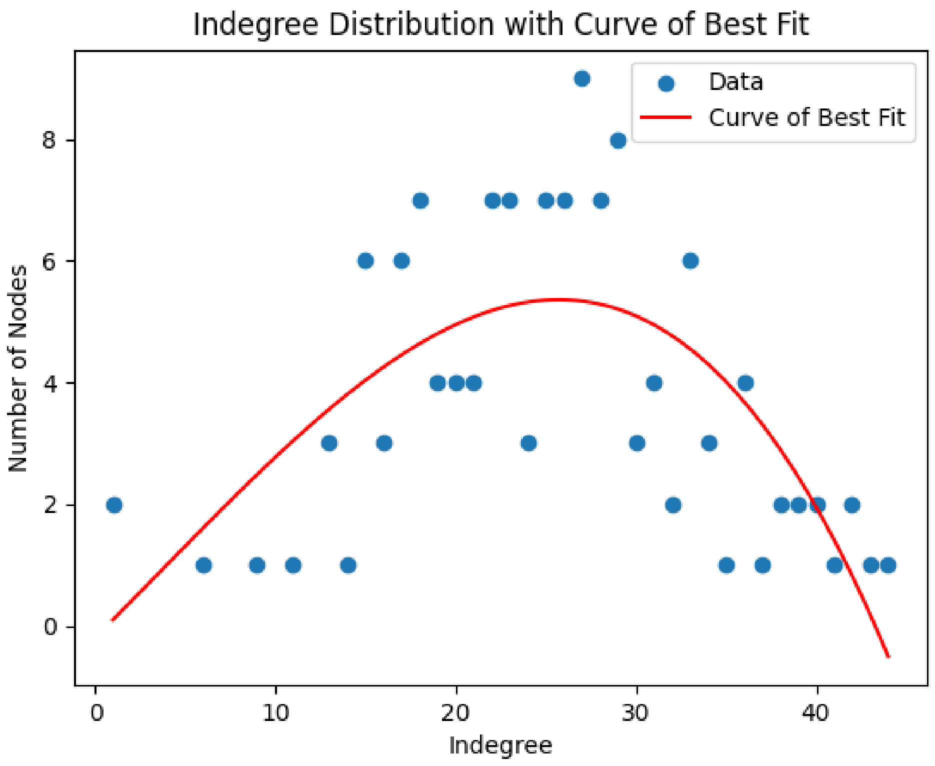
Conclusion

In summary, the observation that the plot of number of nodes versus number of indegrees resembles a normal distribution curve indicates that the network's connectivity pattern does not follow a power law. This insight provides valuable information about the structure and dynamics of the network, guiding further analysis and interpretation.

```python
import networkx as nx
import matplotlib.pyplot as plt
import csv
import numpy as np

# Read the data from the CSV file and create a directed graph
obj = csv.reader(open('data.csv', 'r'))
l = []  # nodelist
e = []  # edgelist
next(obj)
G = nx.DiGraph()
for row in obj:
    b = row[1][0:11].upper()
    l.append(b)
    for i in row[2:]:
        if i == '':
            continue
        e.append((b, i[-11:].upper()))
G.add_nodes_from(l)
G.add_edges_from(e)

# Store the number of inlinks for each node
inlinks = {}
for i in l:
    inlinks[i] = len(list(G.in_edges(i)))

# Make a graph of indegrees on x axis and percentage of nodes with
```

```
    that indegree on y axis
indegree_counts = {}
for i in inlinks.values():
    if i in indegree_counts:
        indegree_counts[i] += 1
    else:
        indegree_counts[i] = 1

x = list(indegree_counts.keys())
y = [i for i in indegree_counts.values()]

# Fit a polynomial curve to the data points
degree = 3  # Choose the degree of the polynomial curve
p = np.polyfit(x, y, degree)
f = np.poly1d(p)

# Generate values for the curve
x_fit = np.linspace(min(x), max(x), 100)
y_fit = f(x_fit)

# Plot the scatter plot and the curve of best fit
plt.scatter(x, y, label='Data')
plt.plot(x_fit, y_fit, color='red', label='Curve of Best Fit')
plt.xlabel('Indegree')
plt.ylabel('Number of Nodes')
plt.title('Indegree Distribution with Curve of Best Fit')
plt.legend()
plt.show()
```

Listing 3: Nature of Curve


Indegree Distribution with Curve of Best Fit
Data
Curve of Best Fit
Number of Nodes
Indegree
0
2
4
6
8
0
10
20
30
40