**A picture containing letter

Description automatically generated**

**CLASS PROJECT 1**

**INTRODUCTION TO BIGDATA AND ANALYTICS**

**By**

**Nikhil Dugyala(G25443567)**

**Rashi Tiwary (G44119910)**

1. **Dataset Introduction:**

The soc-Epinions1\_adj.tsv dataset is a graph of links representing opinions on the SOC-E website. The dataset contains approximately 10 million nodes, with each row of the dataset representing a single edge in the graph connecting two nodes. The format of each row is <node-1>, <node-2>, #Edges, where #Edges is always 1 for each row. Therefore, the third column of the dataset can be removed once it is loaded as a matrix or a data frame.

The dataset presents an interesting opportunity to analyze the structure of the network and gain insights into the opinions expressed on the SOC-E website. By analyzing the connectivity of nodes and identifying clusters of nodes with similar connectivity patterns, we can potentially uncover subcommunities of users with similar opinions. Through this project, we have used R and various data analysis tools to explore and visualize the data in order to gain insights into the structure and properties of the network.

1. **Install the igraph package from one of the CRAN mirrors. Determine how to create a graph and plot. Show the plot in your report.**

* **Loading the dataset**

The first step is to load the dataset into R. We used the read.table function to read the data from the file "soc-Epinions1\_adj.tsv". The header parameter is set to TRUE, which means that the first row of the file contains the column names. The sep parameter is set to '', which means that the columns are separated by tabs.



Figure 2.1

* **Creating a Graph**

After loading the dataset, we converted it into a graph using the igraph library. We extract the columns "X3" and "X1" from the dataset, which represent the nodes of the graph. Figure 1 represents the conversion of the data frame from the dataset given.

We then create a data frame "relations" with these columns and convert it into a graph "plot\_graph" using the graph.data.frame function as shown in Figure 2.3. The unfiltered graph of the dataset is shown in the Figure 2.5.

Text

Description automatically generated

Figure 2.2

Figure 1


Figure 2.3

Figure 1

Text, table

Description automatically generated

Figure 2.4

Shape, circle

Description automatically generated

Figure 2.5

* **Community Detection**

We performed community detection on the graph using the walktrap algorithm provided by the igraph library. This algorithm detects communities in a graph based on random walks.

* **Contracting Vertices**

After detecting communities, we contracted vertices in the same community into a single vertex. This is done using the contract.vertices function provided by the igraph library. We set the vertex attributes to the sum of the weights of the vertices in the same community.

* **Simplifying Edges**

We have simplified the edges of the graph by removing multiple edges between the same vertices. This is done using the simplify function provided by the igraph library.

A picture containing graphical user interface

Description automatically generated

Figure 2.6

* **Induced Subgraph**

Then we have extracted an induced subgraph from the original graph based on the weight of the vertices. We only select vertices with a weight greater than 30. This is done using the induced.subgraph function provided by the igraph library, shown in Figure 2.7 and Figure 2.8.

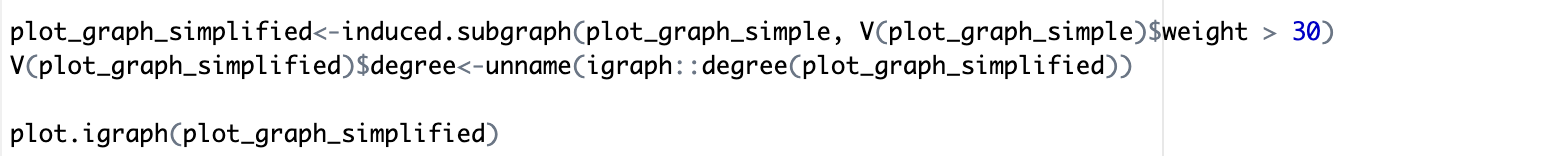


Figure 2.7

Diagram

Description automatically generated

Figure 2.8

* **Degree Distribution**

We plotted the degree distribution of the graph using the hist function provided by R. The degree distribution gives the frequency of the number of nodes with a specific degree.

Chart, histogram

Description automatically generated

Figure 2.9

* **Vertex Removal**

Background pattern

Description automatically generated with medium confidence

Figure 2.10

We removed the vertices with the lowest degree from the graph. Then we have used delete\_vertices function provided by the igraph library to delete the vertices from the graph and we plotted the new graph after removing the vertices with the lowest degree, shown in Figure 2.10.

* **Weight calculation of the graph**: We can add the weight to all the edges and vertices as shown below:

Text

Description automatically generated with medium confidence

Figure 2.11

1. **Applying the functions to the simplified graph**

* **Vertices of the graph**: V(graph\_name)

This function gives us the number of nodes that is present in the graph. The input to this function will be our simplified graph. As shown in the Figure 2.8, after the simplification of the graph, we have 60 nodes.

A picture containing letter

Description automatically generated

Figure 3.1

* **Edges of the graph**: E(graph\_name)

This function gives us the number of edges between the nodes of our simplified graph. As shown in Figure 2.9, we have 562 edges.

Table

Description automatically generated with medium confidence

Figure 3.2

* **Edge density:** Calculating the edge density, we get the information of how strongly the graph is connected. A lower density means that the graph is not that strongly connected and higher density means that the graph is tightly bounded. As shown in Figure 2.10, the density of the graph is quite less, hence the graph is not that heavily connected.

We calculated the edge density without any loops and with the loops in the graph.

Graphical user interface

Description automatically generated with low confidence

Figure 3.3

* **Graph Density:** The graph density of a graph in refers to the ratio of the number of edges in a graph to the maximum possible number of edges in the graph. The density of a graph is a measure of how well connected the vertices are. The gden value ranges between 0 and 1, where a value of 0 indicates a completely disconnected graph and a value of 1 indicates a completely connected graph.

Text

Description automatically generated with low confidence

Figure 3.4

* **Degree of the graph**: Calculates the number of edges of that particular node

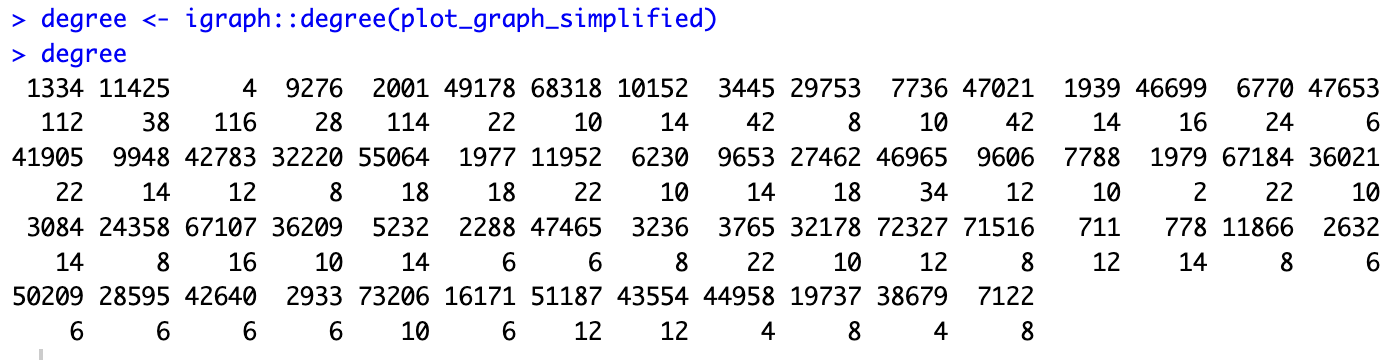


Figure 3.5

* **Gorder**(): gives the number of vertices present in our graph. Figure 2.12 shows that there are 40 vertices in our graph.

A picture containing graphical user interface

Description automatically generated

Figure 3.6

* **Is.simple(graph):** this function is used to check if there are any multiple loops or edges in the graph.

A picture containing graphical user interface

Description automatically generated

Figure 3.7

As we can see, our graph is already in the simplified version.

* **Geodesic:** *A geodesic* is the shortest path between any two nodes in the network. A node has high betweenness if the geodesics between many pairs of other nodes pass through that node. The geodesic distance is a fundamental concept in graph theory and is used to study the structure and properties of graphs.

**Text

Description automatically generated with medium confidence**

Figure 3.8

**3.1. Now, after vertex removal, we will try to work with less number of nodes of the graph.**

* The graph after removing few vertices is shown in Figure 3.8. the edges, vertices and edge density is shown below:

**Text

Description automatically generated**

Figure 3.7

Graphical user interface, text, table

Description automatically generated

Figure 3.8

**Observation**: We can see that picking the first 20 nodes, gives us a smaller number of edges which was expected.

* We can convert the graph generated, into an adjacency matrix as shown below:

Calendar

Description automatically generated

Figure 3.9

* **Betweenness Centrality**: Gives an idea of centrality based on the shortest paths in the graph.

Table

Description automatically generated

Figure 3.10

* **Closeness centrality:** Indicates how close one node is compared to other nodes. It is calculated based as the average of the shortest path length from one node to every other node in the network. As we can see the closeness centrality of the graph is not much, except for the few nodes.

**Graphical user interface, text

Description automatically generated**

Figure 3.11

* **Shortest path:** Gives a shortest path between any 2 vertices. Shown below for initial portion of the graph.

Table

Description automatically generated

Figure 3.12

* If we want to find the number of paths between 2 nodes, then we can multiply the adjacency matrix by itself. Degree distribution of the same is shown in Figure 4.6.

A picture containing calendar

Description automatically generated

Figure 3.13

Chart, histogram

Description automatically generated

Figure 3.14

* **Diameter of the graph:** The diameter of the graph is nothing but the diameter of our graph.

Graphical user interface

Description automatically generated with low confidence

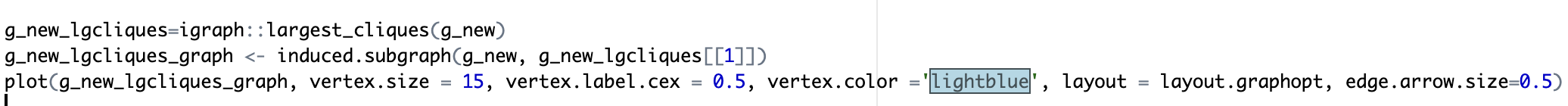
Figure 3.15

1. **Exploring some of the more functions as given in the Rubric:**

* **Largest cliques:** clique is a subset of vertices of an undirected graph such that every two distinct vertices in the clique are adjacent. Our graph contains largest clique of size 8.

A picture containing graphical user interface

Description automatically generated



A picture containing dome, accessory

Description automatically generated

Figure 4.1

* **Central node:** It determines the central scores of the nodes. The node with maximum score is the central node.

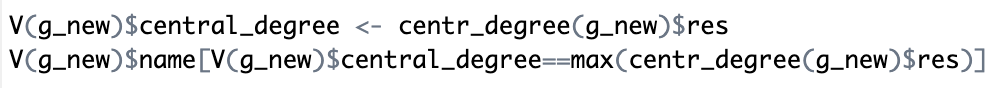




Figure 4.2

* **Ego(s):** Through this function, we find out the neighbors of the particular node.

Table

Description automatically generated

Figure 4.3

* **Power centrality:** It indicates the function of connectivity of the nodes in a neighborhood. It is also known as ‘Bonacich power measure’, which corresponds to the notion that the power of a vertex is recursively defined by the sum of the power of its alters.

A screenshot of a computer

Description automatically generated with medium confidence

Figure 4.4

* **Longest path in the graph:** longest path of the graph is nothing but the longest connected component of the graph. Below is the fraction of code used to calculate the longest path in the graph.

Text

Description automatically generated

A picture containing text, pinwheel, ax, outdoor object

Description automatically generated

Figure 4.5

1. **Observation and Analysis of the Project:**

This project provided a great opportunity to learn and apply various concepts of big data analytics using R. We learned how to load, manipulate, and visualize large datasets using the igraph library in R. Specifically, we learned about the different functions of the igraph library, such as creating a graph, performing community detection, contracting vertices, simplifying edges, and extracting induced subgraphs. We also learned how to calculate various properties of a graph, such as edge density, graph density, and degree distribution, which are useful in understanding the structure and connectivity of a network.

Moreover, this dataset was actually a dataset between users how much they trust each other. This project provided insights into the structure of the network, as we were able to detect communities of users with similar connectivity patterns and uncover subgroups of users with similar opinions. Additionally, we learned how to remove the vertices with the lowest degree from the graph and plot the new graph, which helped in identifying the most important nodes in the network.

Overall, this project helped us develop a better understanding of the principles and tools used in big data analytics, particularly in the context of analyzing and visualizing complex networks. We also learned the importance of exploring data visually and identifying patterns and trends that are not immediately apparent through numerical analysis alone.

**List of Functions:**

* read.table()
* as.data.frame()
* graph.from.data.frame()
* plot()
* edge\_density()
* igraph::E()
* igraph::V()
* degree()
* gorder()
* ego.extract()
* contract.vertices()
* simplify()
* delete.vertices()
* induced.subgraph()
* centr\_clo()
* get.adjacency()
* walktrap.community()
* graph\_from\_adjacency\_matrix()
* diameter()
* count\_triangles()
* articulation\_points()
* mst()
* get\_diameter()
* igraph::largest\_cliques()
* igraph::ego()
* power\_centrality()
* gden()
* geodist()