SOCIAL NETWORK ANALYSIS

IDENTIFICATION OF INFLUENTIAL NODES

Objective :

Finding the top k nodes in the network by using the graph measures and the top k nodes are ordered according to a score derived from all of the discovered network centrality measures.

STEP 1 – DATASET

The Dataset used here is **SBCN-DCTM.** It represents a directed graph. A directed graph is a graph in which the edges have a direction, so they connect one node to another in a specific way. In a directed graph, it is possible for one node to have multiple edges pointing to it from different nodes, but it is not possible for an edge to connect a node to itself.

STEP 2 – APPLYING GRAPH MEASURES

Centrality Measures :

Centrality measures are metrics used to evaluate the importance or influence of a node (such as a person or organization) in a network. These measures help researchers understand the structure and dynamics of a network by quantifying the relative importance of different nodes within the network.

We worked on the following Centrality Measures

- Degree Centrality
- Betweenness Centrality
- Eigenvector centrality
- Closeness centrality
- HITS Algorithm
- Degree prestige
- Rank Prestige
- PageRank
- Percolation Centrality
- Harmonic centrality

Degree Centrality :

Degree centrality is a measure of the importance of a node in a network. It is based on the number of connections that a node has to other nodes in the network. In other words, the degree centrality of a node is the number of edges that are incident on that node.

Id	**Degree_centrality**
9711200	0.12151423917871773
9802150	0.08854449434874882
9802109	0.08148659987167464
9905111	0.06524850698386062

Betweenness Centrality :

Betweenness centrality is a measure of the importance of a node in a network based on the number of times it falls on the shortest path between two other nodes. It reflects the extent to which a node lies on the paths connecting other nodes in the network.

Id	**betweenness_centrality**
9905111	0.11737916518311253
9810008	0.10874169781347577
206223	0.10864132910811432
9803001	0.07092007899495555

Eigenvector centrality :

Eigenvector centrality is a measure of the importance of a node in a network based on the connections of its neighbors. It reflects the idea that a node is important if it is connected to other important nodes, rather than just being connected to many nodes in general. Eigenvector centrality is calculated by solving a system of equations in which the centrality of a node is proportional to the sum of the centralities of its neighbors. This means that if a node is connected to other high-centrality nodes, it will have a high eigenvector centrality itself.

Id	**eigenvector_centrality**
211178	0.3163665700874254
304232	0.2264903005837767

304025	0.15065526472756463
303060	0.14787458509492596

Closeness Centrality :

Closeness centrality is a measure of the importance of a node in a network based on its proximity to other nodes. It reflects the extent to which a node is near or far from other nodes in the network.

Id	**closeness_centrality**
210224	0.14287864615637
210292	0.12992804896987
211178	0.126469548742868
209241	0.12258562326746

HITS Algorithm :

The HITS (Hyperlink-Induced Topic Search) algorithm is a method for ranking web pages based on the structure of their hyperlinks. The HITS algorithm works by identifying two types of pages: authorities and hubs. Authorities are pages that are linked to by many other pages, and are considered to be experts on a particular topic. Hubs are pages that link to many other pages, and are considered to be good starting points for exploring a particular topic.

Id	**Hubs_score**
9711200	0.0258670731977693

9802150	0.0220877000348127
9802109	0.0210810462296095
9905111	0.00799867375631987

Id	authorities_score
9905111	0.00149801672011359
201253	0.000788025360802412
110055	0.000763881894856408
303191	0.000689979514219044

Degree Prestige :

Degree prestige is a term that is sometimes used in the field of social network analysis to refer to the prestige or status associated with having a high degree of connectivity in a social network. In other words, it is the prestige that comes from having many connections to other people within a social network.

Id	Degree_prestige
9711200	0.119145155717881
9802150	0.0876067321455012
9802109	0.0809930408173338
9610043	0.0591777306154681

Rank Prestige :

Rank prestige is a term that is sometimes used in the field of social network analysis to refer to the prestige or status associated with having a high position within a social hierarchy. In other words, it is the prestige that comes from being at the top of a social hierarchy, or from having a high rank within a social group.

Id	rank_prestige
10675	1032
14642	547
12814	532
18761	466

Page Rank :

PageRank is a method for ranking web pages developed by Google as part of its search engine algorithm. It is based on the idea that a web page is important if it is linked to by many other web pages, and that the importance of a page is proportional to the number and quality of the pages that link to it.

Id	Page_rank
207116	0.00123579307157545
303256	0.00107403030145961
304263	0.00105267863295887
9905111	0.00102915778782738

Percolation Centrality :

Percolation centrality is a measure of the importance of a node in a network based on the likelihood that it will be reached by a random walker moving through the network. It reflects the idea that a node is important if it is easy to reach or "percolate" to from other parts of the network.

Id	Percolation_centrality
9905111	0.117379165183156
9810008	0.108741697813515
206223	0.108641329108154
9803001	0.0709200789949796

Harmonic Centrality :

Harmonic centrality is a measure of the importance of a node in a network based on the inverse of the distance from that node to all other nodes in the network. It reflects the idea that a node is important if it is close to many other nodes, and that the importance of a node decreases as its distance to other nodes increases.

Id	Harmonic_centrality
210224	3379.3506132757
210157	3144.84394369823
210292	3138.77616550121
209241	3027.20459540463

Finding top K nodes :

Using minmax normalization to rescale the data between 0 and 1, the rankings of the nodes based on the 13 centrality measurements described above are determined. For various k values, these distinct rankings are intersected with one another. The top k nodes for k value 1000 are the intersection of the 13 sets made up of the nodes with the top 1000 ranks according to each of the centrality metrics.

Ranking the top k nodes :

A score for each node is calculated by adding the all the normalized graph measures. Equal weight is given for each centrality measure here. This score is used to rank the top k nodes.

$$\text{SCORE} = \Sigma \text{ normalized centrality measures}$$

ANP Algorithm :

```
# Import necessary libraries
import numpy as np

# Define a function for calculating the consistency ratio
def consistency_ratio(matrix, n):
  eigenvalues, eigenvectors = np.linalg.eig(matrix)
  eigenvector = eigenvectors[:, np.argmax(eigenvalues)]
  priorities = np.abs(eigenvector / np.sum(eigenvector))
  consistency_index = (np.max(eigenvalues) - n) / (n - 1)
  return consistency_index / 0.58 #random_consistency_index[3] = 0.58

# Define a function for calculating the priority vectors
def priority_vectors(matrix):
  eigenvalues, eigenvectors = np.linalg.eig(matrix)

  eigenvector = eigenvectors[:, np.argmax(eigenvalues)]

  return np.abs(eigenvector / np.sum(eigenvector))


# Define the criteria for comparison

criteria = ["Betweenness Centrality", "Eigenvector Centrality", "Degree Centrality
"]


# Define the comparison matrix

rel_matrix = np.array([[1,3,6],

                              [1/3,1,2],

                              [1/6,1/2,1]])


# Calculate the consistency ratio

consistency_ratio = consistency_ratio(rel_matrix, len(criteria))


# Check if the consistency ratio is acceptable

if consistency_ratio < 0.1:

  # Calculate the priority vectors for the criteria

  priorities = priority_vectors(rel_matrix)

  ndc = list(sorted_degree_centrality.values())/np.linalg.norm(list(sorted_degree_
centrality.values()), ord=np.inf)

  nbc = list(sorted_degree_centrality.values())/np.linalg.norm(list(sorted_between
ness_centrality.values()), ord=np.inf)

  nec = list(sorted_degree_centrality.values())/np.linalg.norm(list(sorted_eigenve
ctor_centrality.values()),  ord=np.inf)


  # Calculate the combined centrality measure

  combined_centrality = np.dot(priorities, np.dot( rel_matrix, [nbc,

                                          nec,

                                          ndc]))


  # Print the results

  combined_cent_dict= {}

  i=0

  for key in keys:

    combined_cent_dict[key] = combined_centrality[i]
    i+=1
else:
  print("The comparison matrix is not consistent. Please try again.")
```

This code defines a set of functions for calculating the consistency ratio and priority vectors of a comparison matrix using the analytic network process (ANP). The first function, consistency_ratio, takes as input a comparison matrix and the number of criteria being considered, and it calculates the consistency ratio of the matrix. The consistency ratio is a measure of how well the matrix is constructed and how consistent the pairwise comparisons are. If the consistency ratio is less than 0.1, the matrix is considered to be consistent and the priorities for the different criteria can be calculated.

The second function, priority_vectors, takes as input a comparison matrix and calculates the priority vectors for the criteria. The priority vector for each criterion is a measure of how important that criterion is relative to the other criteria.

The code also defines a set of criteria for comparison (in this case, three different types of centrality measures for a network) and a comparison matrix that represents the relative importance of the criteria. The code then calculates the consistency ratio for the matrix and, if the consistency ratio is less than 0.1, it calculates the priority vectors and uses them to calculate a combined centrality measure for each node in the network. This combined measure takes into account the relative importance of the different criteria, as determined by the comparison matrix. Finally, the code prints the results of the calculation.

```
from heapq import nlargest


print("Top 100 nodes based on ANP: ")
print("Node\t\tANP\t\tDC\t\tBC\t\tEC")


res = nlargest(100, combined_cent_dict, key = combined_cent_dict.get)
for val in res:
  print(val, combined_cent_dict[val], degree_centrality[val], betweenness_centrali
ty[val], eigenvector_centrality[val])
```

This code prints the top 100 nodes in a network based on their combined centrality measure calculated using the analytic network process (ANP). The code first prints a header line that specifies the columns of information that will be included in the output. The first column is the node ID, and the remaining columns are the ANP centrality measure, the degree centrality, the betweenness centrality, and the eigenvector centrality of each node.

Next, the code uses the **nlargest** function from the **heapq** library to find the 100 nodes with the highest combined centrality measure. The **nlargest** function takes as input a list of values and a key function that specifies how the values should be compared. In this case, the values are the nodes in the network and the key function is the combined centrality measure of each node.

Finally, the code loops through the top 100 nodes and prints the node ID, the combined centrality measure, and the degree, betweenness, and eigenvector centrality measures for each node. This allows the user to see how the combined centrality measure is related to the individual centrality measures for each node.

```
kahp = []
for key in combined_cent_dict:
    kahp.append([key,combined_cent_dict[key]])
fields=["Id","ANP"]
with open(base_path+"Results/ANP-Algorithm.csv",'w') as csvFile:
     csvwriter=csv.writer(csvFile)
     csvwriter.writerow(fields)
     for key in kahp:
         csvwriter.writerow(key)
```

This code writes the results of the calculation of the combined centrality measure using the analytic network process (ANP) to a CSV file. The code first creates a list called **kahp** that contains the node IDs and combined centrality measures for each node in the network. This list is constructed by looping through the **combined_cent_dict** dictionary and appending each key-value pair to the list as a two-element list.

Next, the code opens a CSV file for writing and creates a **csvwriter** object that will be used to write the data to the file. The code then writes the field names (i.e., the column headers) to the file. In this case, the field names are "Id" for the node ID and "ANP" for the combined centrality measure.

Finally, the code loops through the **kahp** list and writes each two-element list (i.e., each node ID and combined centrality measure) to the CSV file using the **csvwriter** object. This allows the results of the ANP calculation to be saved and used for further analysis or visualization.

STEP 3 – K NORMALIZATION

K normalization [for k=1000, 1250, 1500, 1750, 1950, 2250]

The top k ranked nodes are found and they are normalized separately. They are again ranked based on the new values and ordered as done previously.

K value	**Top k nodes count**	**Top k nodes with order**
1000	1	[204054]
1250	3	[204004, 7018, 204054]
1500	5	[204004, 7018, 204054, 202109, 110028]
1750	12	[110108, 107177, 204004, 11115, 7018, 106191, 204054, 108172, 202109, 204146, 9905111, 110028]
1950	28	[110108, 107177, 203101, 204144, 12210, 204004, 11115, 5067, 7018, 8241, 3136, 201253, 7170, 106191, 203018, 204054, 108172, 202109, 204189, 112045, 9905111, 204146, 111093, 203134, 110028, 206038, 205089, 202179]
2200	35	[110108, 107177, 203101, 204144, 12210, 204004, 11115, 6023, 5067, 7018, 8241, 3136, 201253, 7170, 9907086, 8074, 106191, 205101, 110050, 5127, 203018, 204054, 108172, 202109,

		204189, 112045, 3025, 9905111, 204146, 111093, 203134, 110028, 206038, 205089, 202179]

Conclusion :

The networkx package in Python is used to compute the graph metrics, and minmax normalisation is used to standardise the results. The top k nodes in the given network are determined and rated using these graph metrics. To determine the top k nodes, the values of the top k ranked nodes are independently normalised and then intersected. In both instances, the outcome is the same.