Report about

Community detection using k-means clustering based on top influential nodes identified by TOPSIS method

Submitted by:

- LAFIFI Hamza
- AITOUALLANE Abderrahmane

Submitted on:

20 February 2022

Supervised by:

- QAFFOU Issam

Abstract

This report aims to study a complex network by identifying its top-k influential nodes by a proposed method and detecting the communities associated with these nodes using k-means clustering.

The first task focuses on using the proposed method TOPSIS in order to identify the top-k influential nodes. These nodes will be compared to top-k influential nodes identified by the different centrality measures, DC (degree centrality), BC (betweenness centrality), CC (closeness centrality) and EC (eigenvector centrality), using an evaluation model SI (Susceptible, Infected).

The objective of the second task is to use the k-means clustering algorithm in order to determine k clusters based on the top-k influential nodes identified by TOPSIS and to compare its result with other community detection algorithm.

Table of figures

Table of contents

Introduction

In social networks, influential nodes are rare, but they can play a crucial role in effective information spreading. By identifying them, we can get a better control epidemic outbreak.

A simple strategy is to choose top-k ranked nodes as spreaders based on various centrality measures. However, these measures suffer from some limitations and disadvantages: the degree centrality does not take into consideration the global structure of the network, since it only counts how many social connections it has. The lack of applicability of proximity to networks with disconnected components is one of the primary limitations of closeness centrality: two nodes belonging to distinct components do not have a finite distance between them. The problem with the betweenness centrality is that because of a large fraction of nodes in a network do not lie on the shortest path between any two other nodes, they all get the same score of 0. If there are two or more non-identical components, the eigenvector of adjacency matrix will only describe one of them. In such cases, the best course of action is to examine each component separately, that's why the eigenvector centrality cannot be applied to networks that have multiple components. Using each of the centrality measures cause a different ranking of the top influential nodes. In this report we intent to identify the influential nodes by the proposed method, technique for order performance by similarity to ideal solution, as known as TOPSIS. This methos is based on multi-criteria, which are in this project the four centrality measures (DC, BC, CC and EC). And to evaluate its performance, we use the SI (Susceptible-Infected) model to examine the spreading influence of the nodes ranked by TOPSIS method and comparing that influence with the influence of the nodes obtained by the other centrality measures.

Identifying network communities is one of the most significant challenges and it is as important as identifying influential nodes. Communities, at their most basic level, allow us to discover groups of interacting nodes (i.e., objects) and the relationships that exist between them. Communities in social networks, for example, correspond to groups of friends who went to the same school or grew up in the same town; communities in protein interaction networks are functional modules of interacting proteins; and communities in co-authorship networks correspond to scientific disciplines. And we know that clustering is the process of grouping objects together, those in the same group (cluster) are more similar than those in other groups (clusters), this way of working may achieve good results of community detection problems. In this report we aim to adapt a k-means clustering algorithm to large scale problem represented on graphs. Therefore, the goal here is to use the k-means clustering for community detection in a complex network and compare it's result with another community detection algorithm afterwards.

1. Preliminaries

1.1. Network Graph

Network Graph enables you to visualize the relationships between entities, where an entity may be a person, an event, a transaction, a vehicle, or anything else. Entities are represented as nodes, and the relationships between entities are represented as lines that are called links or edges.

1.2. Community

A community can be defined as a subset of nodes that are densely connected to each other and loosely connected to the nodes in the other communities in the same graph. [1]

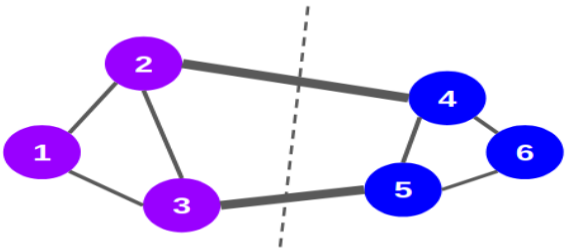


Figure 1: Example of two community in a network graph

1.3. Centrality measures for node influences

1.3.1. The betweenness centrality

Betweenness centrality is a way of detecting the number of times a node acts as a bridge along the shortest path between two other nodes. That means it quantifies the amount of influence a node has over the flow of information in a graph.

$$centrality_{betweenness}(i) = \sum_{s,t \in N} ({}^{\sigma_{s,t}(i)}/_{\sigma_{s,t}})$$

$\sigma_{s,t}$:is the number of shortest paths between nodes s and t.

$\sigma_{s,t}(i)$:is the number of shortest paths between nodes s and t that pass through i.

1.3.2. The closeness centrality

Closeness centrality quantifies how close a node is to all other nodes in the network. In other words, it is a way of detecting nodes that are able to spread information very

efficiently through a graph. It is calculated as the average of the shortest path length from the node to every other node in the network.

$$Centrality_{closeness}(\mathrm{i}) = (|\mathrm{R(i)}|/|\mathrm{N}| - 1) * (|\mathrm{R(i)}|/ \sum_{j \in R(i)} d(i,j)$$

R (i): is the set of all nodes i can reach.

1.3.3. The degree centrality

Degree centrality of a node is simply its degree (the number of edges it has). The higher the degree, the more central the node is. That means it is assigns an importance score based simply on the number of links held by each node.

$$centrality_{degree}(i) = d_i/(|N| - 1)$$

d_i : is the degree node i.

N : is the set of all nodes of the graph.

1.3.4. The eigenvector centrality

Eigenvector Centrality is an algorithm that measures the transitive influence of nodes. Relationships originating from high-scoring nodes contribute more to the score of a node than connections from low-scoring nodes. A high eigenvector score means that a node is connected to many nodes who themselves have high scores. That means a node with few connections could have a very high eigenvector centrality if those few connections were to very well-connected others. Eigenvector centrality allows for connections to have a variable value, so that connecting to some vertices has more benefit than connecting to others.

$$Ax = \lambda x$$

A : is the adjacency matrix of the graph.

λ : is eigenvalue.

By virtue of the Perron–Frobenius theorem, there is a unique solution x, all of whose entries are positive, if λ is the largest eigenvalue of the adjacency matrix. [2]

1.4. TOPSIS

TOPSIS is one of multiple criteria decision-making method that was first introduced by Yoon and Hwang. TOPSIS using the principle that the alternatives selected must have the shortest distance from the positive ideal solution and the farthest from the negative ideal solution from

a geometrical point by using the Euclidean distance to determine the relative proximity of an alternative to the optimal solution. Positive ideal solution is defined as the sum of all the best value that can be achieved for each attribute, while the negative-ideal solution consists of all the worst value achieved for each attribute. TOPSIS into account both the distance of the positive ideal solution and the distance to the negative ideal solution by taking the relative proximity to the positive ideal solution. Based on the comparison of the relative distance, alternative priority order can be achieved. This method is widely used to complete the decision making. This is due to the concept is simple, easy to understand, efficient computation, and has the ability to measure the relative performance of the alternatives decision. [3]

The steps in calculating the TOPSIS method:

- Step 1: Make a decision matrix is normalized.

$$r_{ij} = \frac{X_{ij}}{\sqrt{\sum_{j=1}^{m} X_{ij}^2}}, \qquad i = 1, \dots \dots, m; j = 1, \dots \dots, n.$$

- Step 2: Normalized weighted.
  With the weight $w_j = (w_1, w_2, \dots \dots, w_n)$, where w_j is the weight for j criterion.
  The normalization of weight matrix V, is

$$v_{ij} = w_j * r_{ij}$$

- Step 3: Determining the ideal solution matrix of positive and negative ideal solution by using this formula:

$$A^+ = \{v_1^+, v_2^+, \dots \dots, v_n^+\} = \{(\max v_{ij} | j \in K_b)(\min v_{ij} | j \in K_c)\}$$
$$A^- = \{v_1^+, v_2^+, \dots \dots, v_n^+\} = \{(\min v_{ij} | j \in K_b)(\max v_{ij} | j \in K_c)\}$$

  where K_b is the set of benefit criteria and K_c is the set of cost criteria.

- Step 4: Calculating separation.
  - S^+ is an alternative distance from the positive ideal solution is defined as:

$$S_i^+ = \sqrt{\sum_{j=1}^{n} (v_j^+ - v_{ij})^2}, \qquad i = 1, \dots, m; j = 1, \dots, n.$$

  - S^- is an alternative distance from the negative ideal solution is defined as:

$$S_i^- = \sqrt{\sum_{j=1}^{n} (v_j^- - v_{ij})^2}, \qquad i = 1, \dots, m; j = 1, \dots, n.$$

- Step 5: Calculate the relative closeness to the ideal solution:

$$C_i = \frac{S_i^-}{S_i^- + S_i^+}, \qquad i = 1, \dots, m.$$

- Step 6: Alternative rank.

Alternative C_i sorted from largest value to the smallest value. Alternative with the largest value of C_i the best solution

1.5. SI model

The SI (Susceptible-Infected) model was introduced in 1927 by Kermack.

In this model, during the course of an epidemic, a node is allowed to change its status only from Susceptible (S) to Infected (I).

The model is instantiated on a graph having a non-empty set of infected nodes.

SI assumes that if, during a generic iteration, a susceptible node comes into contact with an infected one, it becomes infected with probability β: once a node becomes infected, it stays infected (the only transition allowed is S→I). [4]

1.6. K-means Clustering

K-means clustering is a type of unsupervised learning, which is used when you have unlabeled data (i.e., data without defined categories or groups). The goal of this algorithm is to find groups in the data, with the number of groups represented by the variable K. The algorithm works iteratively to assign each data point to one of K groups based on the features that are provided. Data points are clustered based on feature similarity. [5]

The results of the K-means clustering algorithm are:

- The centroids of the K clusters, which can be used to label new data
- Labels for the training data (each data point is assigned to a single cluster)

The K-means clustering algorithm uses iterative refinement to produce a final result. The algorithm inputs are the number of clusters K and the data set. The data set is a collection of features for each data point. The algorithm starts with initial estimates for the K centroids, which can either be randomly generated or randomly selected from the data set. The algorithm then iterates between two steps:

a) Data assignment step:

Each centroid defines one of the clusters. In this step, each data point is assigned to its nearest centroid, based on the squared Euclidean distance. More formally, if c_i is the collection of centroids in set C, then each data point x is assigned to a cluster based on

$$\arg\min_{c_i \in C} dist(c_i, x)^2$$

where dist(,) is the standard Euclidean distance. Let the set of data point assignments for each i^{th} cluster centroid be S_i.

b) Centroid update step:

In this step, the centroids are recomputed. This is done by taking the mean of all data points assigned to that centroid's cluster.

$$C_i = \frac{1}{|S_i|} \sum_{x_i \in S_i} x_i$$

1.7. Louvain algorithm

Louvain algorithm is an efficient hierarchical clustering algorithm based on graph theory. Its principle is to make the modularity of community partition result reach the maximum value through continuous iteration of mobile nodes and then obtain the optimal community partition. [6]

The main steps of the Louvain algorithm are as follows:

- Step 1: Initialize the community and set each node as a separate community, namely, community 1: (node1), community 2: (node2), and so on.
- Step 2: Find out all the communities connected to node 1, and calculate the change of modularity after moving node 2 to each neighbor community. Move node 1 to the community, which can increase the modularity to the maximum.
- Step 3: Iterate over all the nodes and execute step 2 until there are no nodes to move and get a layer of community partition.
- Step 4: Merge each community in step 3 into a new node. The relationship between new nodes is the relationship between the original communities. Return to step 1 until all nodes are finally merged into one community. The multilevel community partition is obtained, and the partition with the highest modularity is selected as the final partition result.

1.8. Markov clustering

The Markov Cluster (MCL) Algorithm is an unsupervised cluster algorithm for graphs based on simulation of stochastic flow in graphs. [7]

The motivating idea in MCL is that if you start walking randomly from a node, you are more likely to move around in the same cluster than to cross clusters. This is because by definition clusters are internally dense while being separated by sparse regions. In graph clustering, density and sparsity is defined in terms of the proportion of edge slots that have edges in them.

1.9. Silhouette Method

One of the fundamental steps of an unsupervised learning algorithm is to determine the number of clusters into which the data may be divided. The silhouette algorithm is one of the many algorithms to determine the optimal number of clusters for an unsupervised learning technique.

Silhouette analysis refers to a method of interpretation and validation of consistency within clusters of data. The silhouette value is a measure of how similar an object is to its own cluster (cohesion) compared to other clusters (separation). It can be used to study the separation distance between the resulting clusters. The silhouette plot displays a measure of how close each point in one cluster is to points in the neighboring clusters and thus provides a way to assess parameters like number of clusters visually. [8]

In the Silhouette algorithm, we assume that the data has already been clustered into k clusters by a clustering technique. Then for each data point, we define the following:

C(i) –The cluster assigned to the i_{th} data point.

|C (i)|- The number of data points in the cluster assigned to the i_{th} data point.

a(i) - It gives a measure of how well assigned the i_{th} data point is to it's cluster.

$$a(i) = \frac{1}{|C(i)| - 1} \sum_{C(i), i \neq j} d(i, j)$$

b(i) – It is defined as the average dissimilarity to the closest cluster which is not it's cluster.

$$b(i) = min_{i \neq j} \sum_{j \in C(j)} d(i, j)$$

The silhouette coefficient s(i) is given by:-

$$S(i) = \frac{b(i) - a(i)}{\max (a(i), b(i))}$$

We determine the average silhouette for each value of k and for the value of k which has the maximum value of s(i) is considered the optimal number of clusters for the unsupervised learning algorithm.

1.10. Elbow Method

The Elbow Method is one of the most popular methods to determine this optimal value of k. we are actually varying the number of clusters K from 1 – N, we assume that the data has already been clustered into k clusters by a clustering technique. For each value of K, we are calculating WCSS (Within-Cluster Sum of Square). WCSS is the sum of squared distance between each point and the centroid in a cluster. When we plot the WCSS with the K value, the plot looks like an Elbow. As the number of clusters increases, the WCSS value will start to decrease. WCSS value is largest when K = 1. When we analyze the graph, we can see that the graph will rapidly change at a point

and thus creating an elbow shape. From this point, the graph starts to move almost parallel to the X-axis. The K value corresponding to this point is the optimal K value or an optimal number of clusters.

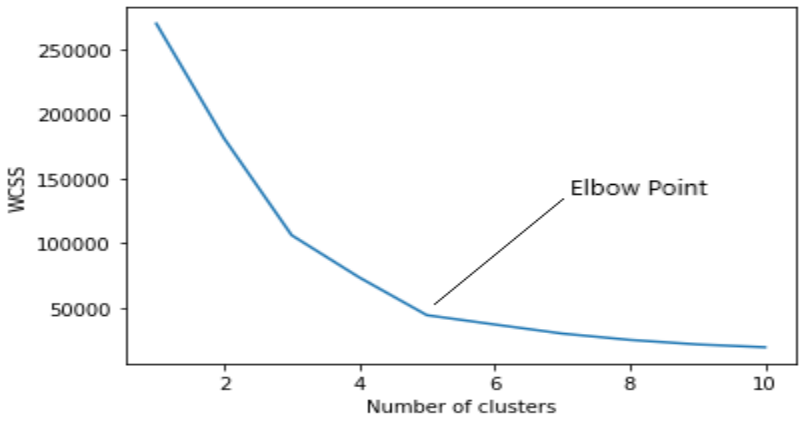


Figure 2: Plot of the WCSS by the K value if N=10

2. Software implementation

In this section we present the software environment including the programming language and its libraries and the integrated development environment (IDE) that we have used in this project.

2.1. programming language

- Python

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built-in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed. [9]

2.2. Used libraries

2.2.1. Math

The Python Math Library provides us access to some common math functions and constants in Python, which we can use throughout our code for more complex mathematical computations. The library is a built-in Python module, therefore you don't have to do any installation to use it. In this article, we will be showing example usage of the Python Math Library's most commonly used functions and constants. [10]

2.2.2. Numpy

NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more. [11]

Figure 3: Numpy library python

2.2.3. Pandas

Pandas is a Python library for data analysis. Started by Wes McKinney in 2008 out of a need for a powerful and flexible quantitative analysis tool, pandas has grown into one of the most popular Python libraries. It has an extremely active community of contributors.

Pandas is built on top of two core Python libraries Matplotlib for data visualization and NumPy for mathematical operations. Pandas acts as a wrapper over these libraries, allowing you to access many of matplotlib's and NumPy's methods with less code. For instance, pandas.plot() combines multiple matplotlib methods into a single method, enabling you to plot a chart in a few lines. [12]

Figure 4: Pandas library python

2.2.4.NetworkX

The NetworkX Package is a Python library for studying graphs and networks. It provides tools for the creation, manipulation, and study of dynamic and complex network structures. With NetworkX, we can load and store networks in many data formats, generate many types of random and classic networks, analyze network structure, build network models, design new network algorithms, draw networks, and much more. In this tutorial, we will learn how to use NetworkX to create graphs and study networks. [13]

Figure 5: NetworkX library python

2.2.5. Matplotlib

Matplotlib is a python library used to create 2D graphs and plots by using python scripts. It has a module named pyplot which makes things easy for plotting by providing feature to control line styles, font properties, formatting axes etc. It supports a very wide variety of graphs and plots namely - histogram, bar charts, power spectra, error charts etc. It is used along with NumPy to provide an environment that is an effective open-source alternative for MatLab.

Figure 6: Matplotlib library python

2.2.6.NDlib

NDlib is a Python software package that allows to describe, simulate, and study diffusion processes on complex networks. [14]

Figure 7: NDlib library python

2.2.7. CDlib

CDlib is a Python software package that allows to extract, compare and evaluate communities from complex networks. [15]

Figure 8: CDlib library python

2.2.8. Sklearn

Scikit-learn (Sklearn) is the most useful and robust library for machine learning in Python. It provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction via a consistence interface in Python. This library, which is largely written in Python, is built upon NumPy, SciPy and Matplotlib. [16]

Figure 9: Sklearn library python

2.3. Environment (IDE)

- **Jupyter notebook**

The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations, and narrative text. Its uses include data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more.

Jupyter Notebook (formerly IPython Notebooks) is a web-based interactive computational environment for creating Jupyter notebook documents. The “notebook” term can colloquially make reference to many different entities, mainly the Jupyter web application, Jupyter Python web server, or Jupyter document format depending on context. [17]

3. Experimental analysis

3.1. Data

In order evaluate the performance of the suggested method in this paper, a Facebook network dataset is used to show its efficacy. This dataset consists of 'circles' (or 'friends lists') from Facebook. Facebook data was collected from survey participants using a Facebook app. The dataset includes node features (profiles), circles, and ego networks. It is available at [Facebook-dataset].

Dataset statistics	
Nodes	4039
Edges	88234
Nodes in largest WCC	4039 (1.000)
Edges in largest WCC	88234 (1.000)
Nodes in largest SCC	4039 (1.000)
Edges in largest SCC	88234 (1.000)
Average clustering coefficient	0.6055
Number of triangles	1612010
Fraction of closed triangles	0.2647
Diameter (longest shortest path)	8
90-percentile effective diameter	4.7

Figure 10: Facebook dataset statistics

3.2. TOPSIS method: Implementation & Evaluation

3.2.1. Implementation

i. Constructing the network

Importing the necessary libraries:

```
import matplotlib.pyplot as plt
import networkx as nx
import pandas as pd
import numpy as np
import math
```

After that, we create the graph G and we fill it with data (Nodes and edges) of file, its name is "FacebookNet".

```
G = nx.Graph()#Create an empty graph G
data = open("facebookNet.txt","r")#read data from file FacebookNet
lines=data.readlines()
for line in lines:
    line_split=line.split(" ")
    G.add_edge(int(line_split[0]),int(line_split[1]))#Add two nodes if not exist and an edge between them.
```

Visualizing the network of the "FacebookNet" dataset:

```
nx.draw_networkx(G, node_color='blue',with_labels=False, edgecolors='black', node_size=35)#Draw the graph G
```

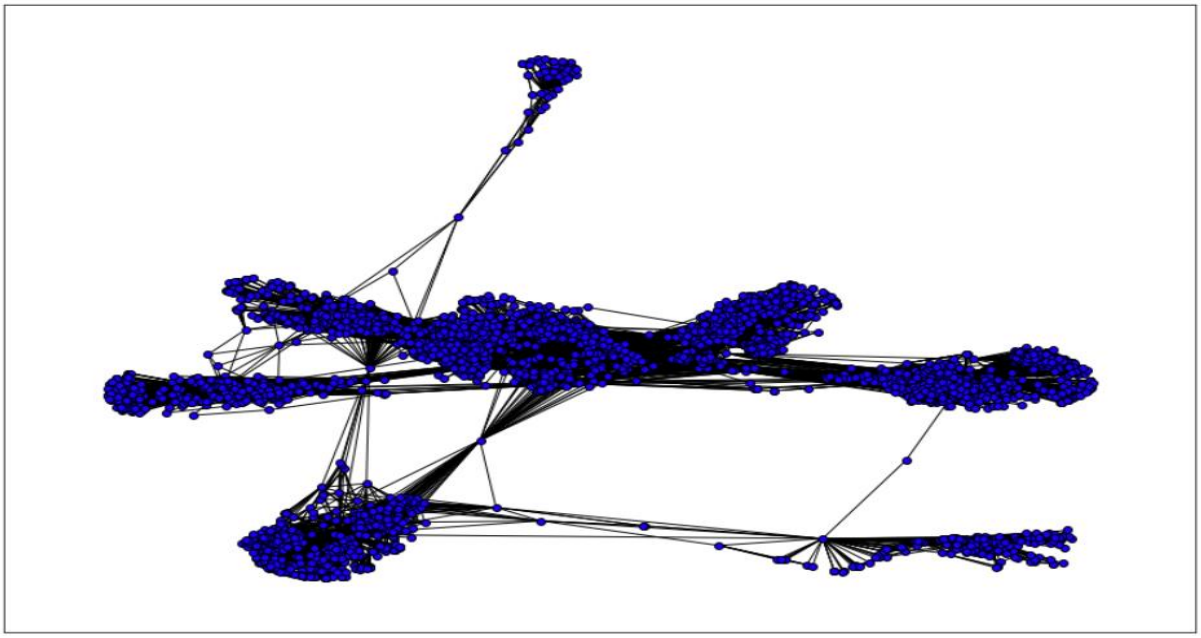


Figure 11: Graph of Facebook dataset

ii. Compute the centrality measures

Here, we calculate the four centrality measures (degree centrality, closeness centrality, betweenness centrality, eigenvector centrality):

```
deg_centrality = nx.degree_centrality(G)#Compute the degree centrality for nodes of the graph G
```

```
close_centrality = nx.closeness_centrality(G)#Compute the closeness centrality for nodes of the graph G
```

```
#Compute the betweenness centrality for nodes of the graph G
bet_centrality = nx.betweenness_centrality(G, normalized = True, endpoints = False)
```

```
#Compute the eigenvector centrality for nodes of the graph G
eig_centrality=nx.eigenvector_centrality(G)
```

Constructing the decision matrix by putting each node with its centrality measures in a single DataFrame called “table”:

```
table=pd.DataFrame()#create dataframe
for noued in G:
    #add the name of each node with its measures centrality
    table=table.append({'Node':str(noued), 'DC':deg_centrality[noued], 'BC':bet_centrality[noued],
                        'CC':close_centrality[noued], 'EC':eig_centrality[noued]},ignore_index=True)
table.to_csv("Data/nodesByCentralityMeasures.csv",index=False)#save our dataframe in file csv
table
```

	Node	DC	BC	CC	EC
0	0	0.085934	1.463059e-01	0.353343	3.391796e-05
1	1	0.004210	2.783274e-06	0.261376	6.045346e-07
2	2	0.002476	7.595021e-08	0.261258	2.233461e-07
3	3	0.004210	1.685066e-06	0.261376	6.635648e-07
4	4	0.002476	1.840332e-07	0.261258	2.236416e-07
...	...	...	...	...	...
4034	4034	0.000495	0.000000e+00	0.183989	2.951270e-10
4035	4035	0.000248	0.000000e+00	0.183980	2.912901e-10
4036	4036	0.000495	0.000000e+00	0.183989	2.931223e-10
4037	4037	0.000991	7.156847e-08	0.184005	2.989233e-10
4038	4038	0.002229	6.338922e-07	0.184047	8.915175e-10

iii. Normalization of the centrality measures

```
def Normalize(table,weights):
    Sum_DC_pow=(table['DC']**2).sum()#
    Sum_BC_pow=(pow(table['BC'],2)).sum()
    Sum_CC_pow=(table['CC']**2).sum()
    Sum_EC_pow=(table['EC']**2).sum()
    result=table.copy()
    result['DC']=(result['DC']/math.sqrt(Sum_DC_pow))*weights[0]
    result['BC']=(result['BC']/math.sqrt(Sum_BC_pow))*weights[1]
    result['CC']=(result['CC']/math.sqrt(Sum_CC_pow))*weights[2]
    result['EC']=(result['EC']/math.sqrt(Sum_EC_pow))*weights[3]
    return result
tableNorm = Normalize(table,[0.2,0.3,0.3,0.2])
```

The function “Normalize” take the decision matrix and a list contains the weights of each centrality measures as input, and it implement step1, and step 2 in TOPSIS method (Make a decision matrix is normalized, Normalized weighted.) and it returns a normalized decision matrix as a DataFrame.

iv. Calculate the relative closeness

```
def  calculate_relative_closeness(table):
    solutions=pd.DataFrame()

    #Positive ideal solution
    max_BC=table['BC'].max()
    max_CC=table['CC'].max()
    max_DC=table['DC'].max()
    max_EC=table['EC'].max()

    #Negative ideal solution
    min_BC=table['BC'].min()
    min_CC=table['CC'].min()
    min_DC=table['DC'].min()
    min_EC=table['EC'].min()


    for i in range(len(table)):
        #Calculating separation
        S_positive=math.sqrt(pow(max_DC-table['DC'][i],2) + pow(max_BC-table['BC'][i],2)
                             + pow(max_CC-table['CC'][i],2) + pow(max_EC-table['EC'][i],2))

        S_negative=math.sqrt(pow(min_DC-table['DC'][i],2) + pow(min_BC-table['BC'][i],2)
                             + pow(min_CC-table['CC'][i],2) + pow(min_EC-table['EC'][i],2))
        #Calculate the relative closeness to the ideal solution
        C = S_negative/(S_negative+S_positive)

        solutions = solutions.append({'Node':table['Node'][i],
                                      'S+':S_positive,'S-':S_negative,'Topsis':C},ignore_index=True)

    return solutions

result = calculate_relative_closeness(tableNorm)
```

The function above calculates the relative closeness or the TOPSIS score which has been already explained as the last three steps of TOPSIS method : Step 3(determining the ideal solution matrix of positive and negative), Step 4(Calculating separation) and Step 5 (Calculate the relative closeness).

v. Identifying the top influential nodes depending on Ci (relative closeness)

Finally, we identify the top influential nodes (top 10 for example) by ranking, from largest value to the smallest value, the nodes based on its relative closeness score:

```
#we sort the nodes by their relative closeness and we put the top 10 in variable "top10Topsis"
top10Topsis = result[['Node','Topsis']].sort_values('Topsis',ascending=False).head(10)
#we maintain the top 10 in the file "top10NodesTopsis.csv"
top10Topsis.to_csv("Data/top10NodesTopsis.csv",index=False)
top10Topsis
```

	Node	Topsis
107	107	0.913277
351	1684	0.695566
352	1912	0.496063
1821	3437	0.488865
0	0	0.304321
571	1085	0.297379
1843	698	0.231106
1710	567	0.193646
58	58	0.169464
350	428	0.132923

3.2.2. Evaluation

In this section, we are going to evaluate the efficiency of TOPSIS method, to do so, we compare the influence of the top 10 influential nodes obtained by TOPSIS method and the other centrality measures using the SI model.

The following table shows the top 10 influential nodes by different centrality measures and TOPSIS method:

	DCN	BCN	CCN	ECN	TopsisN
0	107	107	107	1912	107
1	1684	1684	58	2266	1684
2	1912	3437	428	2206	1912
3	3437	1912	563	2233	3437
4	0	1085	1684	2464	0
5	2543	0	171	2142	1085
6	2347	698	348	2218	698
7	1888	567	483	2078	567
8	1800	58	414	2123	58
9	1663	428	376	1993	428

Figure 12: The top-10 ranked nodes by TOPSIS, DC, BC, CC and EC

As mentioned above, the SI model is used as comparison tool. In this model every node has two discrete states: Susceptible or Infected and able to spread the information (disease) to other susceptible nodes with a probability of □. In each Implementation top-10 nodes that either appear in the top-10 list by TOPSIS method or other four centrality measures will be infected, then the information spreads in the network until the whole network get infected. The less time it takes to infect the whole network the more the nodes have a better influence in the network.

In this paper we used two different values of □ (Infection probability), 0.3 and 1.

i. $\beta = 0.3$

Starting by $\beta = 0.3$, which is a more realistic value, in each iteration a set of nodes will be infected randomly by a probability of 0.3. I order to increase the precision of the results, the algorithm will be repeated 10000 times. Thus, the variation (standard deviation) and the mean of each iteration will be calculated.

```
def SI_model(Nodes):
    outputlst = []
    # Model selection
    model = ep.SIModel(G)

    # Model Configuration
    cfg = mc.Configuration()
    cfg.add_model_parameter('beta', 0.3)
    cfg.add_model_initial_configuration('Infected', Nodes)
    model.set_initial_status(cfg)

    output = pd.DataFrame(columns = ['Iteration', 'nbrSuscepible', 'nbrInfected'])
    for i in range(N):
        # Simulation execution
        iteration = model.iteration()
        output.loc[len(output.index)] = [iteration['iteration'], iteration['node_count'][0], iteration['node_count'][1]]
        if iteration['node_count'][1] == N:
            break
    outputlst.append(output['Iteration'].tolist())
    outputlst.append(output['nbrSuscepible'].tolist())
    outputlst.append(output['nbrInfected'].tolist())

    return outputlst
```

The above function takes as input 10 nodes, and infect them in the network, and calculate the number of the infected nodes in each iteration with the probability of □=0.3 until the whole network get infected and returns a list contains the number of iteration and the number of susceptible and infected nodes in each iteration.

```
DClst = []
BClst = []
CClst = []
EClst = []
TOPSISlst = []
iteration = 10000 #Number of iterations
for i in range(iteration):
    DClst.append(SI_model(top10ByDC))
    BClst.append(SI_model(top10ByBC))
    CClst.append(SI_model(top10ByCC))
    EClst.append(SI_model(top10ByEC))
    TOPSISlst.append(SI_model(top10ByTopsis))
```

The code above will execute the SI_model function 10000 times as mentioned before.

The following function Std_mean() takes as input a list of all the 10000 iterations, and each iteration contains the number of susceptible and infected nodes by iteration, and eventually it will return the mean and the standard deviation of each iteration. For example, in iteration 0, 10 nodes are infected, and in the next iteration a random number of nodes will be infected depending on the influence of the initial nodes and the infection probability and this number will be appended to a list, so this list will contain the number of all of the nodes infected in iteration 1 of all the 10000 times.

```
def Std_mean(lis):
    dc = {}
    sd = []
    mean = []
    for i in range(iteration):
        for j in range(len(lis[i][0])):
            dc[lis[i][0][j]]=[]
    for i in range(iteration):
        for j in range(len(lis[i][0])):
            dc[lis[i][0][j]].append(lis[i][2][j])
    for i in dc:
        sd.append(np.std(dc[i]))
        mean.append(np.mean(dc[i]))
    return mean, sd
```

Here we gather the means and the standard deviations of different iterations and different centrality measures and TOPSIS method.

```
DCmean, DCsd = Std_mean(DClst)
BCmean, BCsd = Std_mean(BClst)
CCmean, CCsd = Std_mean(CClst)
ECmean, ECsd = Std_mean(EClst)
TOPSISmean, TOPSISsd = Std_mean(TOPSISlst)
```

- **Visualization**

The code bellow aims to visualize the output of the obtained results, this visualization contains four plots, each one of them represents a comparison in term of number of infected nodes (y-axis) by iteration (x-axis) between the TOPSIS method and the other centrality measures.

The mean represents the center of the data in each iteration. The standard deviation describes the variation of the data in each iteration, we used standard deviation multiply by 3 because it represents almost the entire data (99.73% of the data).

```
fig, ax = plt.subplots(2, 2,figsize=(15, 9))
ax[0, 0].set_title('topsis VS degree centrality')
ax[0, 0].plot(list(range(len(TOPSISmean))), TOPSISmean, label='Topsis')
ax[0, 0].errorbar(list(range(len(TOPSISmean))), TOPSISmean, yerr=[x * 3 for x in TOPSISsd], fmt ='.k')
ax[0, 0].plot(list(range(len(DCmean))) , DCmean, label='DC')
ax[0, 0].errorbar(list(range(len(DCmean))), DCmean, yerr=[x * 3 for x in DCsd], fmt ='.k')
ax[0, 0].legend()

ax[1, 0].set_title('topsis VS closeness centrality')
ax[1, 0].plot(list(range(len(TOPSISmean))), TOPSISmean, label='Topsis')
ax[1, 0].errorbar(list(range(len(TOPSISmean))), TOPSISmean, yerr=[x * 3 for x in TOPSISsd], fmt ='.k')
ax[1, 0].plot(list(range(len(CCmean))) , CCmean, label='CC')
ax[1, 0].errorbar(list(range(len(CCmean))) , CCmean, yerr=[x * 3 for x in CCsd], fmt ='.k')
ax[1, 0].legend()

ax[0, 1].set_title('topsis vs eigenvector centrality')
ax[0, 1].plot(list(range(len(TOPSISmean))), TOPSISmean, label='Topsis')
ax[0, 1].errorbar(list(range(len(TOPSISmean))), TOPSISmean, [x * 3 for x in TOPSISsd], fmt ='.k')
ax[0, 1].plot(list(range(len(ECmean))) , ECmean, label='EC')
ax[0, 1].errorbar(list(range(len(ECmean))) , ECmean, yerr=[x * 3 for x in ECsd], fmt ='.k')
ax[0, 1].legend()

ax[1, 1].set_title('topsis vs betweeness centrality')
ax[1, 1].plot(list(range(len(TOPSISmean))), TOPSISmean, label='BC')
ax[1, 1].errorbar(list(range(len(TOPSISmean))), TOPSISmean, yerr=[x * 3 for x in TOPSISsd], fmt ='.k')
ax[1, 1].plot(list(range(len(BCmean))) , BCmean, label='Topsis')
ax[1, 1].errorbar(list(range(len(BCmean))) , BCmean, yerr=[x * 3 for x in BCsd], fmt ='.k')
ax[1, 1].legend()

for ax in ax.flat:
    ax.set(xlabel='Iteration', ylabel='nbr of infected nodes')
fig.suptitle('Visualisation')
plt.show()
```

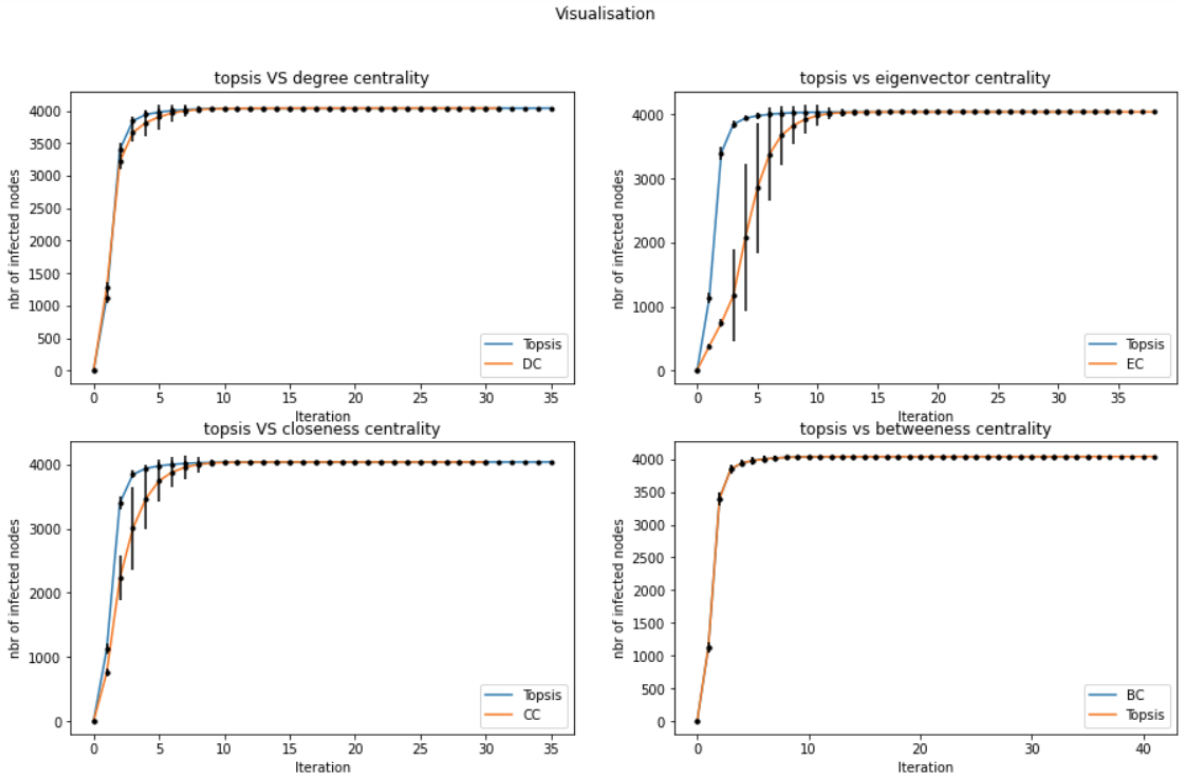


Figure 13: SI Visualization beta=0.3

The above visualizations show that the TOPSIS method is the first to infect the entire network compared to EC and the last compared to DC, BC and CC. Thus, it is the better than the EC and inferior in term of performance than DC, BC and CC.

The problem faced here is the instability of the output result, with every new execution of the same code we may get a different result of the comparison. This issue might be because of the randomization of the next chosen nodes to be infected, and in order to solve this problem we tried to augment the number of iterations to 100000 which took more than three days to complete the execution, and still the results are instable which means we need to increase the number of iterations more and more which will take a long time with the available hardware.

Another proposed solution is to change the value of the infection probability β to 1, this will make assure the same results no matter how many times we repeat the execution.

ii. β = 1

The infection probability equal to 1 means that in each iteration all of the susceptible nodes which directly connected to the infected nodes will be infected and that will lead to a faster spread of information. This kind of spread is not real in fact, because nothing could spread with the probability of 1. However, we can ignore that fact because our goal here is to compare in term of performance.

The function bellow is almost the same as the last SI_model described function except of the value of β.

```
def SI_model(Nodes):
    outputlst = []
    # Model selection
    model = ep.SIModel(G)

    # Model Configuration
    cfg = mc.Configuration()
    cfg.add_model_parameter('beta', 1)
    cfg.add_model_initial_configuration('Infected', Nodes)
    model.set_initial_status(cfg)

    output = pd.DataFrame(columns = ['Iteration', 'nbrSuscepible', 'nbrInfected'])
    for i in range(N):
        # Simulation execution
        iteration = model.iteration()
        output.loc[len(output.index)] = [iteration['iteration'], iteration['node_count'][0], iteration['node_count'][1]]
        if iteration['node_count'][1] == N:
            break

    return output
```

Applying the SI_model function for each method:

```
DC = SI_model(top10ByDC)
BC = SI_model(top10ByBC)
CC = SI_model(top10ByCC)
EC = SI_model(top10ByEC)
TOPSIS = SI_model(top10ByTopsis)
```

Here is an example of the output of SI_model applied on the closeness centrality:

```
In [33]: print(CC)
```

Out[33]:

	Iteration	nbrSuscepible	nbrInfected
0	0	4029	10
1	1	1893	2146
2	2	768	3271
3	3	142	3897
4	4	0	4039

- **Visualization**

The code bellow represents the same comparison of the last visualization after changing the value of β to 1:

```
fig, axs = plt.subplots(2, 2, figsize=(15, 9))
axs[0, 0].plot(TOPSIS['Iteration'], TOPSIS['nbrInfected'], label="Topsis")
axs[0, 0].plot(DC['Iteration'], DC['nbrInfected'], label="DC")
axs[0, 0].set_title("DC vs Topsis")
axs[0, 0].legend()

axs[0, 1].plot(TOPSIS['Iteration'], TOPSIS['nbrInfected'], label="Topsis")
axs[0, 1].plot(BC['Iteration'], BC['nbrInfected'], label="BC")
axs[0, 1].set_title("BC vs Topsis")
axs[0, 1].legend()

axs[1, 0].plot(TOPSIS['Iteration'], TOPSIS['nbrInfected'], label="Topsis")
axs[1, 0].plot(CC['Iteration'], CC['nbrInfected'], label="CC")
axs[1, 0].set_title("CC vs Topsis")
axs[1, 0].legend()

axs[1, 1].plot(TOPSIS['Iteration'], TOPSIS['nbrInfected'], label="Topsis")
axs[1, 1].plot(EC['Iteration'], EC['nbrInfected'], label="EC")
axs[1, 1].set_title("EC vs Topsis")
axs[1, 1].legend()

for ax in axs.flat:
    ax.set(xlabel='Iteration', ylabel='nbr of infected nodes')
```

The bellow plots show a comparison between TOPSIS and the other centrality measures for β=1.

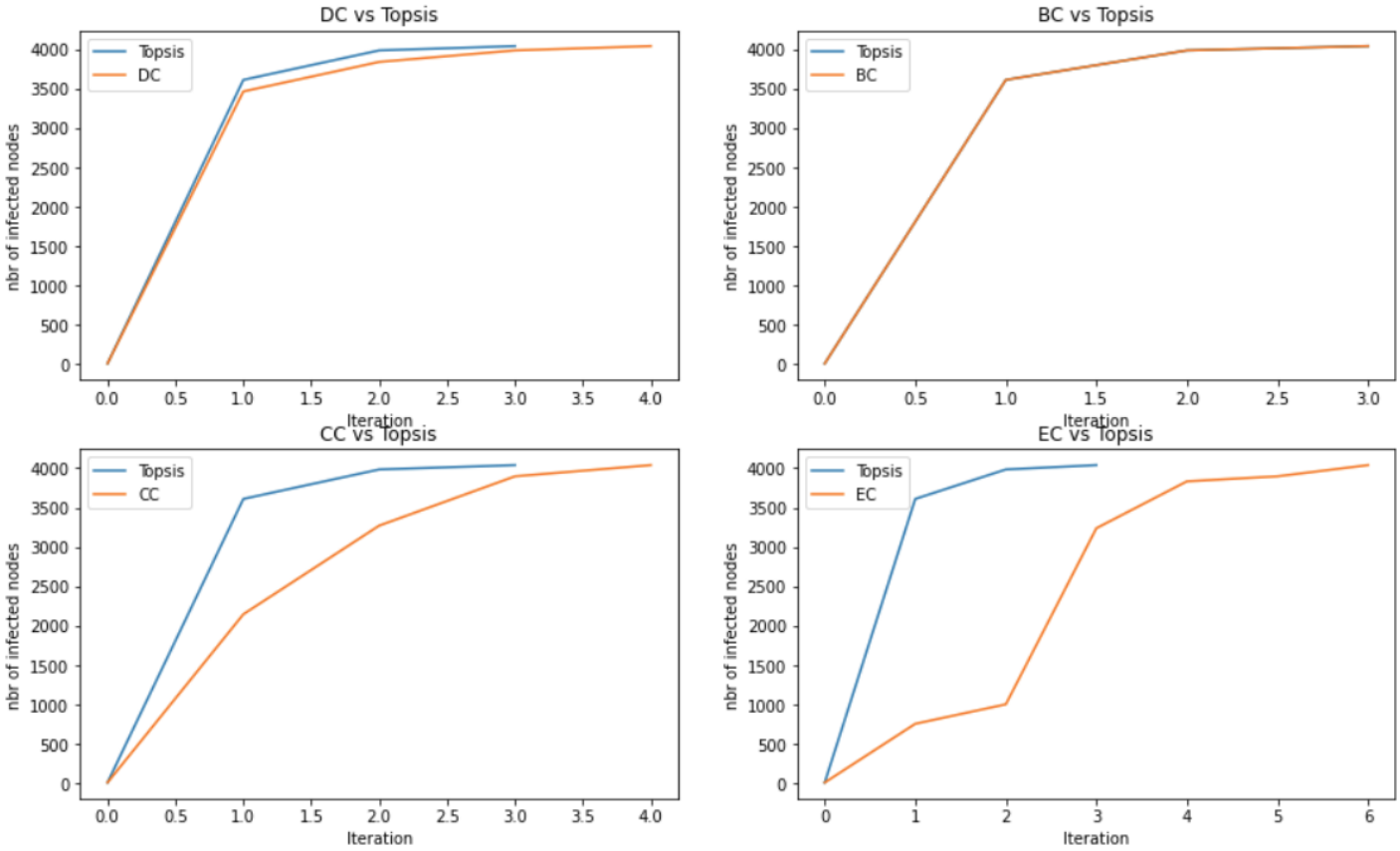


Figure 14:SI Visualization beta=1

We can easily conclude that TOPSIS method performs better than DC, CC and EC, and as good as BC which is normal because the top-10 nodes of TOPSIS method and BC are the same.

3.3. Comparative analysis for k-means clustering algorithm for community detection based on top-10 nodes by TOPSIS

3.3.1. K-means implementation

Detecting the community structure could be expressed by real networks is an important step toward understanding them. Many of the algorithms presented thus far, one of them is the k-means clustering which known by its great efficiency and accuracy. We put two k-means approaches to the test, first one is based on centrality measures of each node, the second one is by using the adjacency matrix of the network.

In the next two section we going to explain each one of the used approaches.

i. K-means based on centrality measures

Importing the necessary packages:

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import networkx as nx
from sklearn.cluster import KMeans
```

The following lines of code are for preparing the training data which is the centrality measures of each node:

In [34]:
```
#Read our decision Matrix
data = pd.read_csv("decisionMatrix.csv")
data.head(10)
#preparing the training data
trainingData = data[["DC","BC","CC","EC"]]
trainingData
```

Out[34]:

	DC	BC	CC	EC
0	0.085934	1.463059e-01	0.353343	3.391796e-05
1	0.004210	2.783274e-06	0.261376	6.045346e-07
2	0.002476	7.595021e-08	0.261258	2.233461e-07
3	0.004210	1.685066e-06	0.261376	6.635648e-07
4	0.002476	1.840332e-07	0.261258	2.236416e-07
...	...	...	...	...
4034	0.000495	0.000000e+00	0.183989	2.951270e-10
4035	0.000248	0.000000e+00	0.183980	2.912901e-10
4036	0.000495	0.000000e+00	0.183989	2.931223e-10
4037	0.000991	7.156847e-08	0.184005	2.989233e-10
4038	0.002229	6.338922e-07	0.184047	8.915175e-10

4039 rows × 4 columns

Now, we prepare the top-10 influential nodes by TOPSIS in order to initialize the centroids of the k-means algorithm:

In [35]:
```
#read the top 10 influential nodes
influential = pd.read_csv("Data/rankedNodes.csv")
inf10 = influential.head(10)
inf10
```

Out[35]:

	Node	DC	BC	CC	EC	Node.1	Topsis
0	107	0.258791	0.480518	0.459699	2.606940e-04	107	0.913277
1	1684	0.196137	0.337797	0.393606	7.164260e-06	1684	0.695566
2	1912	0.186974	0.229295	0.350947	9.540696e-02	1912	0.496063
3	3437	0.135463	0.236115	0.314413	9.531613e-08	3437	0.488865
4	0	0.085934	0.146306	0.353343	3.391796e-05	0	0.304321
5	1085	0.016345	0.149015	0.357852	3.164082e-06	1085	0.297379
6	698	0.016840	0.115330	0.271189	1.116876e-09	698	0.231106
7	567	0.015602	0.096310	0.328881	9.932295e-06	567	0.193646
8	58	0.002972	0.084360	0.397402	5.898120e-04	58	0.169464
9	428	0.028479	0.064309	0.394837	5.990065e-04	428	0.132923

In [36]:
```
top10_init = inf10[["DC","BC","CC","EC"]]
top10_init = top10_init.to_numpy()
top10_init
```

Out[36]:
```
array([[2.58791481e-01, 4.80518079e-01, 4.59699454e-01, 2.60693991e-04],
       [1.96136701e-01, 3.37797450e-01, 3.93605615e-01, 7.16425979e-06],
       [1.86973749e-01, 2.29295340e-01, 3.50947332e-01, 9.54069615e-02],
       [1.35463101e-01, 2.36115357e-01, 3.14412520e-01, 9.53161293e-08],
       [8.59336305e-02, 1.46305921e-01, 3.53342667e-01, 3.39179617e-05],
       [1.63447251e-02, 1.49015092e-01, 3.57851826e-01, 3.16408207e-06],
       [1.68400198e-02, 1.15330450e-01, 2.71188717e-01, 1.11687629e-09],
       [1.56017831e-02, 9.63103312e-02, 3.28880925e-01, 9.93229483e-06],
       [2.97176820e-03, 8.43602059e-02, 3.97401831e-01, 5.89812015e-04],
       [2.84794453e-02, 6.43090624e-02, 3.94837196e-01, 5.99006469e-04]])
```

After preparing the data, now we're going straight to k-means implementation:

In [55]:
```
#initialize the parameters
k_clusters = 10
kmeans = KMeans(n_clusters = k_clusters, init = top10_init, max_iter = 20000, n_init = 1)
```

In [56]:
```
#start the trainning
label = kmeans.fit_predict(trainingData)
```

Here is the output:

```
In [57]: #convert to DataFrame
         lab = pd.DataFrame(label, columns=['Cluster'])
```

```
In [58]: community = pd.concat([data, lab], axis=1)
         community.head(20)
```

Out[58]:

	Node	DC	BC	CC	EC	Cluster
0	0	0.085934	1.463059e-01	0.353343	3.391796e-05	4
1	1	0.004210	2.783274e-06	0.261376	6.045346e-07	7
2	2	0.002476	7.595021e-08	0.261258	2.233461e-07	7
3	3	0.004210	1.685066e-06	0.261376	6.635648e-07	7
4	4	0.002476	1.840332e-07	0.261258	2.236416e-07	7
5	5	0.003219	2.205964e-06	0.261308	1.183322e-06	7

Grouping nodes by their clusters:

```
In [59]: gk = community.groupby('Cluster')
         gk.first()
```

Out[59]:

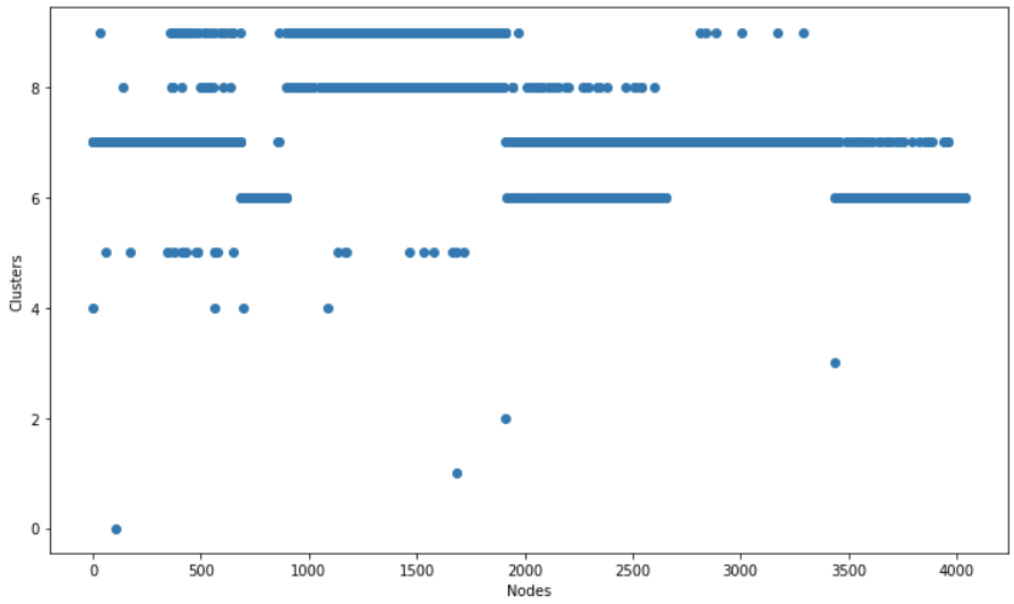
Cluster	Node	DC	BC	CC	EC
0	107	0.258791	0.480518	0.459699	2.606940e-04
1	1684	0.196137	0.337797	0.393606	7.164260e-06
2	1912	0.186974	0.229295	0.350947	9.540696e-02
3	3437	0.135463	0.236115	0.314413	9.531613e-08
4	0	0.085934	0.146306	0.353343	3.391796e-05
5	58	0.002972	0.084360	0.397402	5.898120e-04
6	3980	0.014611	0.024820	0.225448	4.722356e-08
7	1	0.004210	0.000003	0.261376	6.045346e-07
8	136	0.032937	0.026870	0.294186	3.894375e-03
9	34	0.001238	0.003602	0.303313	4.074185e-06

```
In [42]: #Group all Nodes of communities in lists
         comm = []
         for i in range(10):
             comm.append(gk.get_group(i)['Node'].tolist())
```

The scatterplot bellow shows the distribution of the nodes by clusters:

```
In [63]: #Visualize Nodes by Clusters
         plt.scatter(community['Node'], community['Cluster'])
         plt.ylabel("Clusters")
         plt.xlabel("Nodes")
         plt.show
```

```
Out[63]: <function matplotlib.pyplot.show(close=None, block=None)>
```



➢ Visualization

```
colors=['red','blue','yellow','green','lightblue','brown','grey','purple','pink','orange']
for i in range(10):
    nx.draw(Graph, pos=sp1, nodelist=comm[i], node_color= colors[i],with_labels=False,node_size=size[i], edgecolors='black')
```

Nodes in the same cluster have the same color, the big nodes are the top-10 nodes:

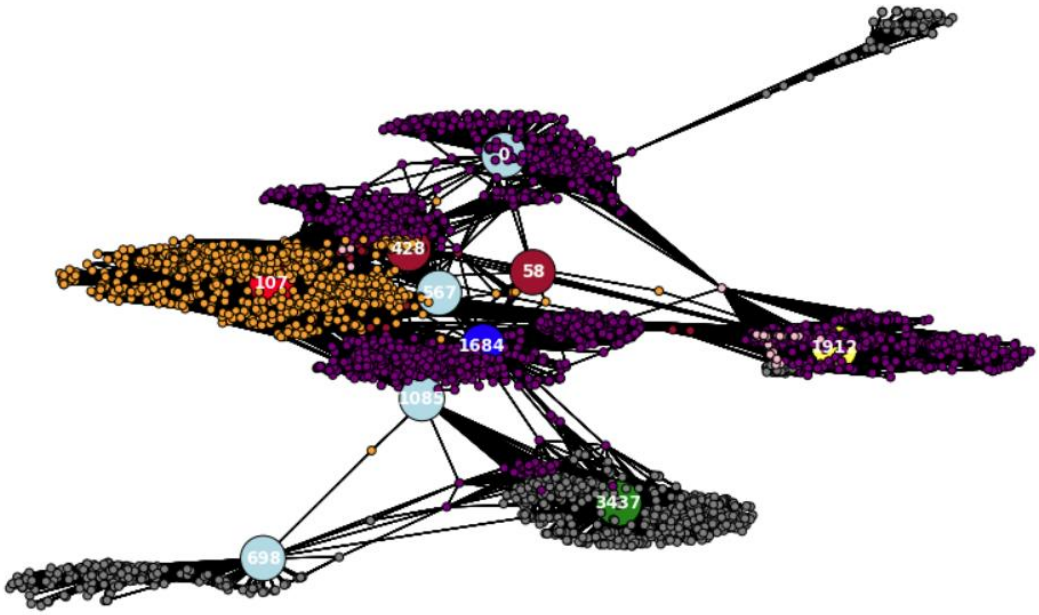


Figure 15: K-means clustering using centrality measures on Facebook graph with centroids initialization

And here is the visualization without initializing the k-means centroids with the top-10 nodes:

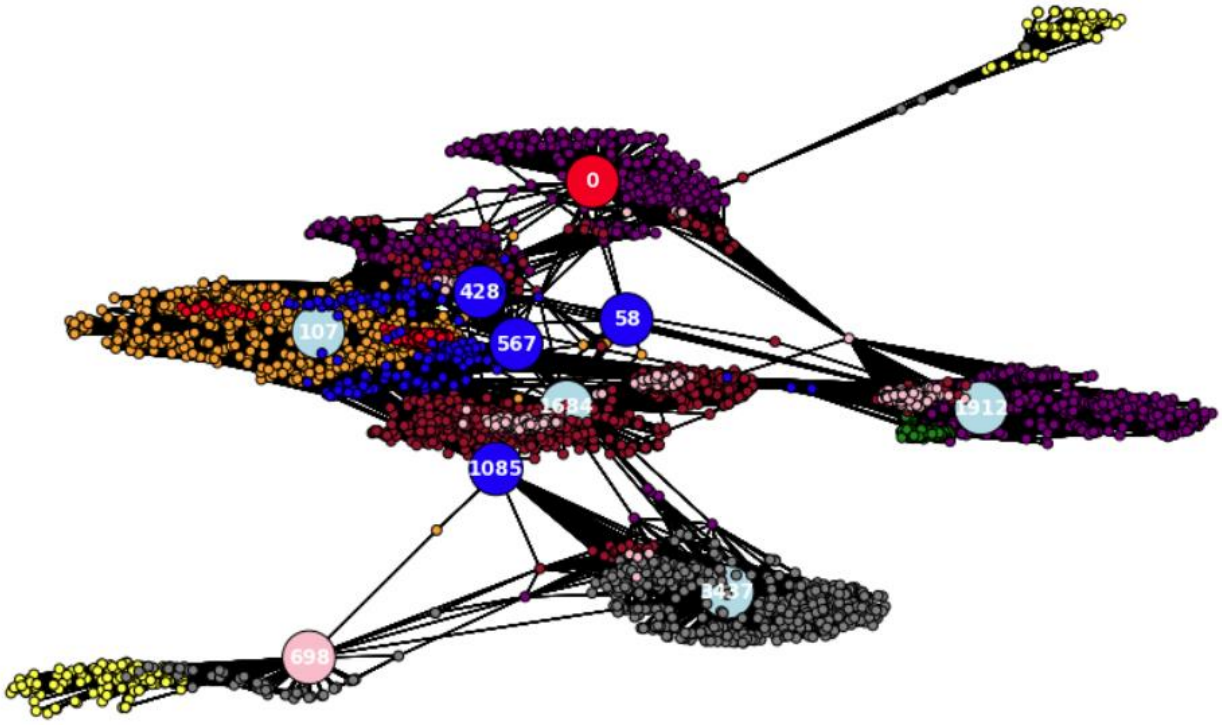


Figure 16: K-means clustering using centrality measures on Facebook graph without centroids initialization

ii. K-means based on adjacency matrix

After importing the libraries and building the network as shown below:

```
G = nx.Graph()
data = open("facebookNet.txt","r")
lines=data.readlines()
for line in lines:
    line_split=line.split(" ")
    G.add_edge(int(line_split[0]),int(line_split[1]))
```

We're going straight to construct the adjacency matrix of the network, and of the top-10 nodes:

```
In [117]: # construction of the adjacency matrix
          edge_mat= nx.to_numpy_array(G, nodelist=[i for i in range(4039)])
          edge_mat

Out[117]: array([[0., 1., 1., ..., 0., 0., 0.],
                 [1., 0., 0., ..., 0., 0., 0.],
                 [1., 0., 0., ..., 0., 0., 0.],
                 ...,
                 [0., 0., 0., ..., 0., 0., 0.],
                 [0., 0., 0., ..., 0., 0., 0.],
                 [0., 0., 0., ..., 0., 0., 0.]])
```

In [118]:
```
# adjacency matrix of top10Nodes
top10NodesTopsis = pd.read_csv("Data/top10NodesTopsis.csv")
top10NodesTopsis.rename(columns = {'Node':'TopsisN'}, inplace = True)
top10NodesTopsis = top10NodesTopsis['TopsisN'].tolist()
edge_mat10 = edge_mat[top10NodesTopsis]
```

Out[118]:
```
array([[1., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [1., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
```

After preparing the data, next step is to start the k-means training:

In [119]:
```
#initialize the parameters
k_clusters = 10
results = []
algorithms = {}
model = KMeans(n_clusters=k_clusters, n_init=1, max_iter=20000, init = edge_mat10)
```

In [125]:
```
#start the trainning
model.fit(edge_mat)
results = list(model.labels_)
```

Nodes by their clusters:

In [129]:
```
Clusters = pd.DataFrame(results)
```

In [136]:
```
#Display each Node by its Cluster
Nodes = []
for Node in G:
    Nodes.append(Node)
Nodes = pd.DataFrame([i for i in range(4039)], columns = ['Node'])
Clusters = pd.DataFrame(results, columns = ['Cluster'])
nodeByCluster = pd.concat([Nodes, Clusters], axis = 1)
nodeByCluster
```

Out[136]:

	Node	Cluster
0	0.0	4
1	1.0	8
2	2.0	8
3	3.0	8
4	4.0	8

Here, we're just grouping nodes by their clusters and splitting them into lists:

```
In [123]: #Display first Node of each Cluster
          gk = nodeByCluster.groupby('Cluster')
          gk.first()
```

Out[123]:

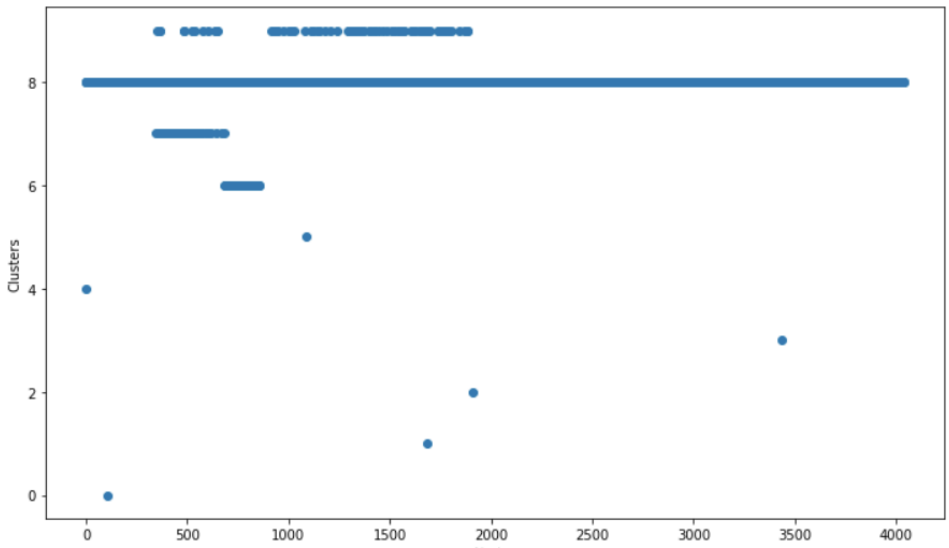
Cluster	Node
0	107
1	536
2	767
3	2600
4	0
5	1599
6	1200
7	348
8	1
9	2814

```
In [124]: #Group all Nodes of communities in lists
          community = []
          for i in range(10):
              community.append(gk.get_group(i)['Node'].tolist())
```

Here is the scatterplot that shows the distribution of the nodes by clusters:

```
In [138]: #Visualize Nodes by Clusters
          plt.scatter(nodeByCluster['Node'], nodeByCluster['Cluster'])
          plt.ylabel("Clusters")
          plt.xlabel("Nodes")
          plt.show
```

```
Out[138]: <function matplotlib.pyplot.show(close=None, block=None)>
```



➢ Visualization

```
colors=['orange','blue','yellow','lightblue','brown','grey','green','pink','purple','red']
for i in range(10):
    nx.draw_networkx(G, pos=sp1, nodelist=community[i], node_color= colors[i],with_labels=False,node_size=30, edgecolors = 'black
```

- With initialization of centroids by top-10 nodes:

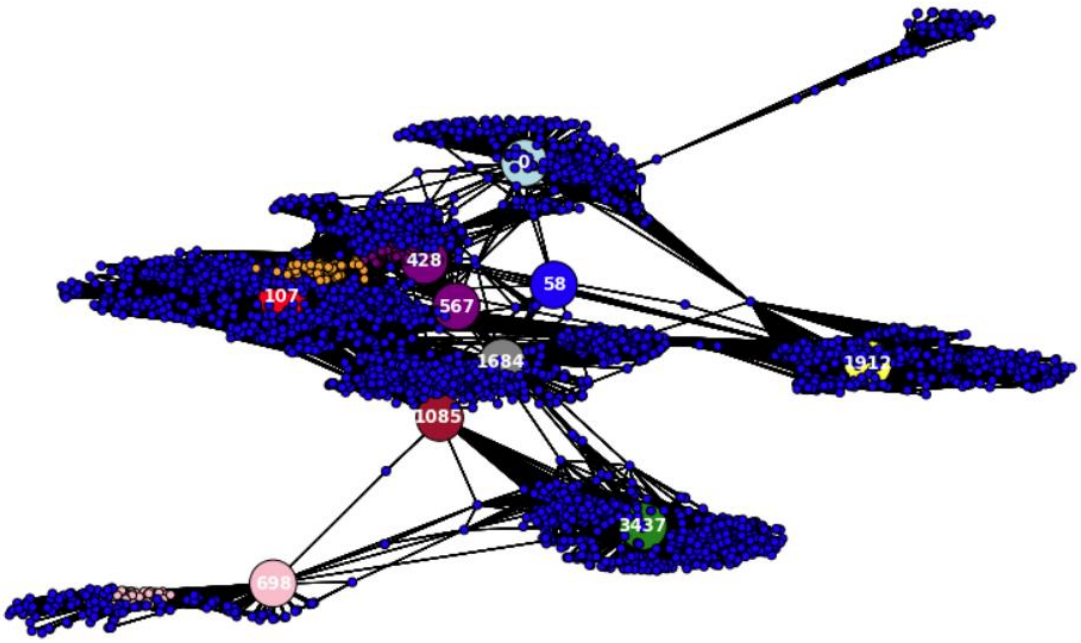


Figure 17: K-means clustering using adjacency matrix on Facebook graph with centroids initialization

- Without initialization of centroids:

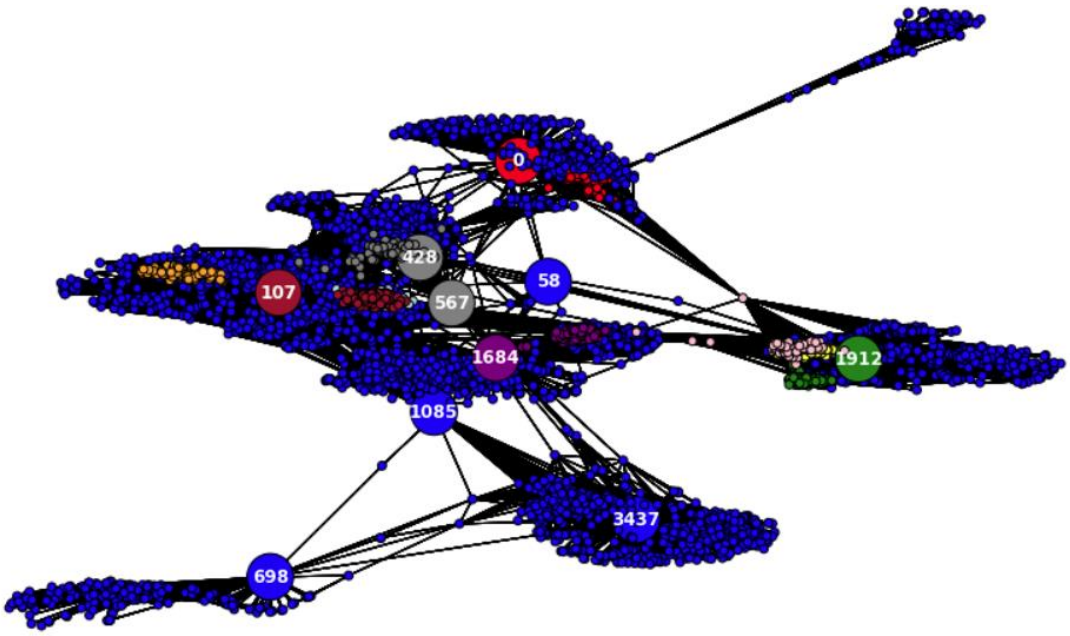


Figure 18: K-means clustering using adjacency matrix on Facebook graph without centroids initialization

3.3.2. Community detection algorithms

In this section we aim to use some community detection algorithms on our network in order to compare their output results with what we obtained by the k-means algorithm. To do so, we chose two different community detection algorithms: Louvain and Markov algorithms.

i. Community detection algorithm: Markov

Importing the necessary packages:

```
import markov_clustering as mc
import networkx as nx
import matplotlib.pyplot as plt
```

Reading the graph of the network and constructing the adjacency matrix:

```
In [2]: # Load network data
        G = nx.Graph()
        data = open("facebookNet.txt","r")
        lines=data.readlines()
        for line in lines:
            line_split=line.split(" ")

            G.add_edge(int(line_split[0]),int(line_split[1]))
```

```
In [18]: # Build adjacency matrix
         edg_mat = nx.to_numpy_array(G)
```

The code below aims to define the size of each node. The top-10 nodes will get bigger size:

```
#the loop below aims to increase the size of the top 10 nodes in TOPSIS
list1=[107,1684,1912,3437,0,1085,698,567,58,428]
size=[]
for i in G:
    if i in list1:
        size.append(1000)
    else:
        size.append(35)
```

We run the Markov clustering algorithm on the adjacency matrix of our network:

```
# Run MCL algorithm
result = mc.run_mcl(edg_mat)
clusters = mc.get_clusters(result)
```

➢ **Visualization**

```
plt.figure(figsize=(15,8))
mc.draw_graph(edge_mat, clusters, node_size=size, with_labels=False,edgecolors='black')
```

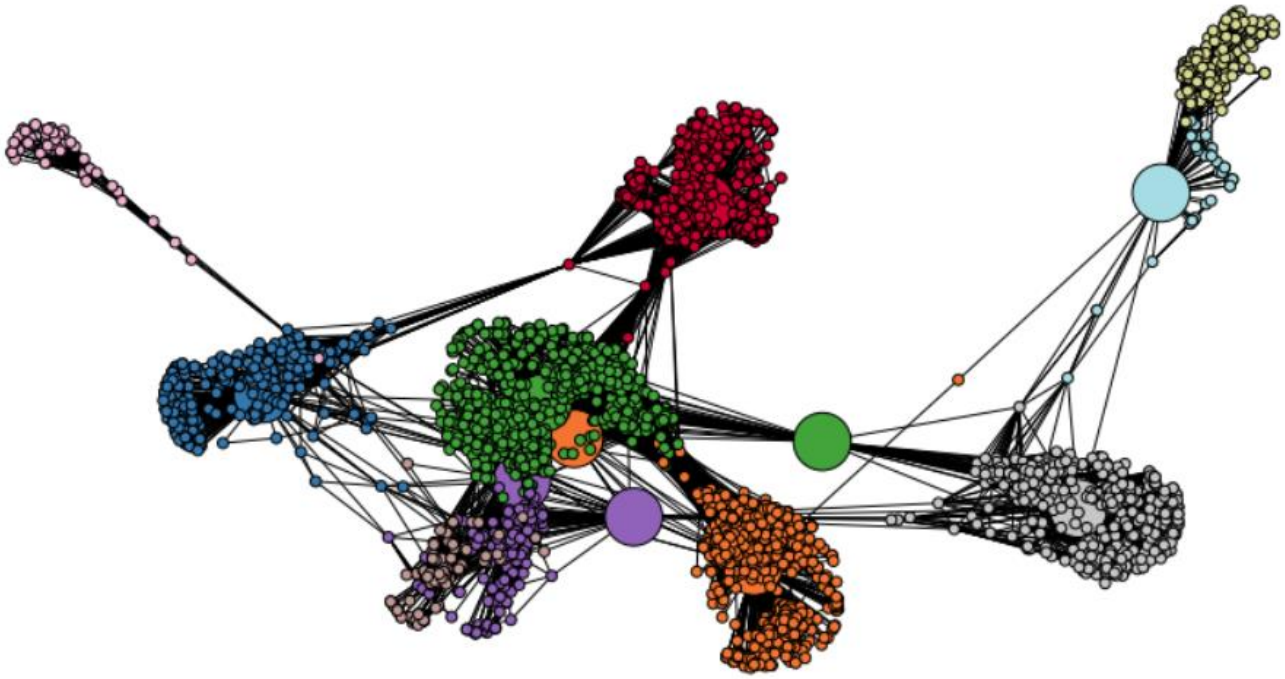


Figure 19: Community detection by Markov algorithm on Facebook graph

ii. Community detection algorithm: Louvain algorithm

Importing the necessary packages:

```
from cdlib import algorithms, viz
import matplotlib.pyplot as plt
import networkx as nx
```

Reading the data and applying the Louvain algorithm:

```
G =nx.read_edgelist("facebookNet.txt")
#implement the louvain algorithm
coms = algorithms.louvain(G)
```

➢ **Visualization:**

```
#draw graph
viz.plot_network_clusters(G,coms,node_size=35,figsize=(15,8),plot_labels=False)
```

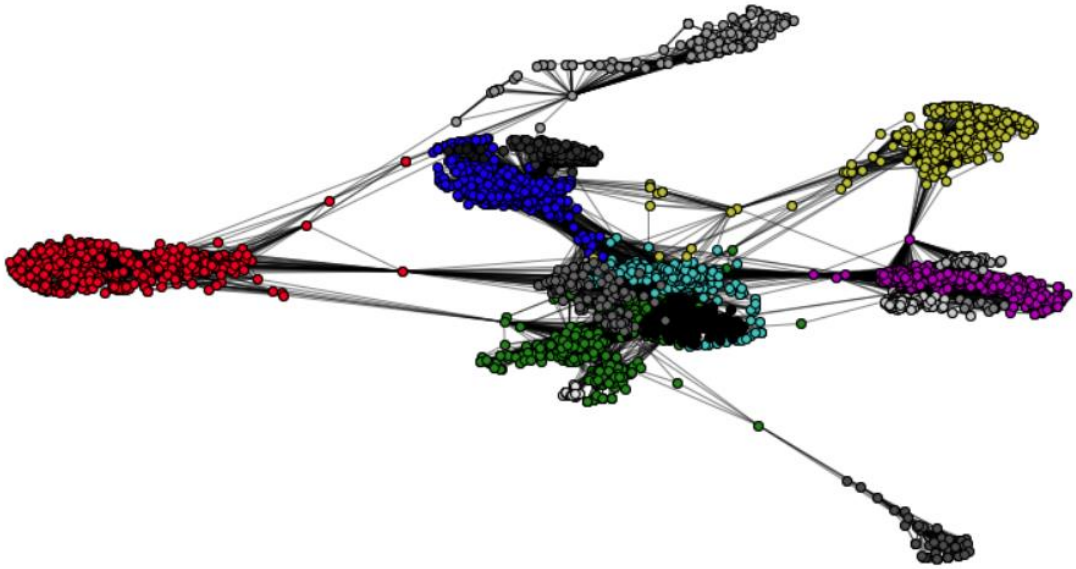


Figure 20: Community detection by Louvain algorithm on Facebook graph

3.3.3. Discussion

In this section we discuss all the results obtained above, starting with the k-means using centrality measures and adjacency matrix.

By using the centrality measures as nodes features for k-means input data, we can notice as shown in Figure 15 that the top-10 nodes have been isolated in separate 6 clusters as the table below shows:

	Node	Topsis	Cluster
107	107	0.913277	0
351	1684	0.695566	1
352	1912	0.496063	2
1821	3437	0.488865	3
0	0	0.304321	4
571	1085	0.297379	4
1843	698	0.231106	4
1710	567	0.193646	4
58	58	0.169464	5
350	428	0.132923	5

Figure 21: Top-10 nodes by its clusters based on k-means (centrality measures)

And the interesting part in this clustering is that some of the top-10 nodes are the only nodes in their cluster like in the case of 107, 1684, 1912 and 3437, and other four of them are gathered in one cluster, and this is the case of 0, 1085, 567 and 698, the two other cluster of the top-10 nodes are grouped with other few nodes in the same cluster as showed in the figure bellow.

```
Cluster[ 0 ] =  [107]
Cluster[ 1 ] =  [1684]
Cluster[ 2 ] =  [1912]
Cluster[ 3 ] =  [3437]
Cluster[ 4 ] =  [0, 1085, 567, 698]
Cluster[ 5 ] =  [58, 171, 348, 414, 428, 353, 376, 420, 475, 483, 484, 563, 566, 580, 651, 1136, 1165, 1171, 1465, 1534, 1577,
1666, 1687, 1718]
```

Figure 22: Clusters by nodes by k-means (centrality measures)

What we can conclude from here is that the k-means clustering using the centrality measures cluster the nodes based on their influence in the network, that's why we can notice that the more the nodes have more influence in the network the more they are isolated in other clusters by themselves.

We can conclude almost the same thing in the FIG, although we did not initialize the centroids with the top-10 nodes.

In the other hand, using the adjacency matrix gave us somehow a closer clustering from what we got by using the centrality measures, the figure below shows this:

```
Cluster[ 0 ] =  [107]
Cluster[ 1 ] =  [1684]
Cluster[ 2 ] =  [1912]
Cluster[ 3 ] =  [3437]
Cluster[ 4 ] =  [0]
Cluster[ 5 ] =  [1085]
Cluster[ 6 ] =  [686, 687, 688, 693, 694, 695, 696, 697, 698, 701, 703, 705, 706, 708, 709, 711,
```

Figure 23: Clusters by nodes by k-means (adjacency matrix)

However, comparing these results with other community detection algorithms is very challenging considering the complexity of this network. Accordingly, it's very difficult to evaluate the performance of the k-means algorithm in detection communities. Thus, in the next sections we will use a dataset of a smaller network (Zachary's karate club).

3.4. Dynamic k-means

Before using the k-means clustering technique, we must declare the number of clusters. It is unable to calculate the optimal number of clusters. K-means divides the data set into the number of clusters we specify in advance. For us, determining the optimal number of clusters is likewise a difficult challenge. We can't simply look at the data set and determine how many partitions we need. Consequently, a lot methods are proposed to solve this issue, in this section we're going to use some of them.

To implement these methods, we're going to apply them on two different datasets, our Facebook network dataset and Zachary dataset.

3.4.1. Elbow method

The elbow method chooses an optimal value of k depending on the distance between data points and their associated clusters using the sum of squared distance (SSE). We'd pick a k value where the SSE starts to flatten out and an inflection point appears. The method's name comes from the fact that this graph resembles an elbow when displayed.

In the following we're going to work with the two datasets and for each network graph we will use its adjacency matrix and its centrality measures as nodes characteristics.

i. Facebook network dataset

- Adjacency matrix

Starting by importing the necessary packages and constructing the network graph:

```
from sklearn import cluster
import networkx as nx
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
import pandas as pd
from kneed import KneeLocator
```

```
G = nx.Graph()
data = open("facebookNet.txt","r")
lines=data.readlines()
for line in lines:
    line_split=line.split(" ")
    G.add_edge(int(line_split[0]),int(line_split[1]))
```

Now, we apply the elbow method on our graph:

```
K =range(1, 20)
X=nx.to_numpy_array(G, nodelist=[i for i in range(4039)])
#we tried to execute elbow method 10 times to show and clarify that it is not stable in this case
for i in range(10):
    inertias=[]
    for k in K:
        # Building and fitting the model
        kmeanModel = KMeans(n_clusters=k)
        kmeanModel.fit(X)
        inertias.append(kmeanModel.inertia_)

    kn = KneeLocator(K, inertias, curve='convex', direction='decreasing')
    print(kn.knee)
```

```
5
4
7
4
4
5
7
6
5
4
```

As mentioned above, the problem with working with the adjacency matrix was the instability of defining the optimal k, after running the code multiple times we got four different values, which means that n_init=10 (n_init represents the number of time k-means algorithm will be run with different centroid seeds) is not enough to get to the steady state of choosing the optimal k value. So, to get there we need to increase n_init value more and more and that is almost impossible because of the complexity of the adjacency matrix. Thus, we will focus on using the centrality measures instead.

- **Centrality measures**

Preparing the training data:

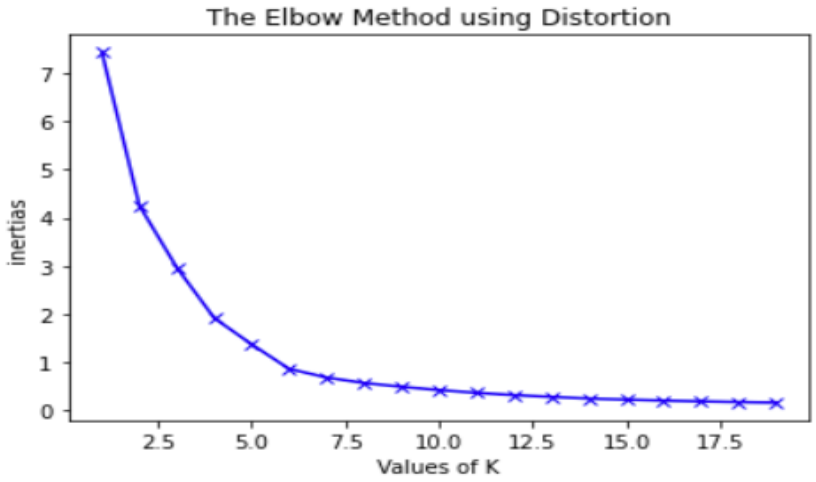
	DC	BC	CC	EC
0	0.085934	1.463059e-01	0.353343	3.391796e-05
1	0.004210	2.783274e-06	0.261376	6.045346e-07
2	0.002476	7.595021e-08	0.261258	2.233461e-07
3	0.004210	1.685066e-06	0.261376	6.635648e-07
4	0.002476	1.840332e-07	0.261258	2.236416e-07
...	...	...	...	...
4034	0.000495	0.000000e+00	0.183989	2.951270e-10
4035	0.000248	0.000000e+00	0.183980	2.912901e-10
4036	0.000495	0.000000e+00	0.183989	2.931223e-10
4037	0.000991	7.156847e-08	0.184005	2.989233e-10
4038	0.002229	6.338922e-07	0.184047	8.915175e-10

4039 rows × 4 columns

Applying the elbow method

```
K = range(1, 20)
inertias=[]

for k in K:
    # Building and fitting the model
    kmeanModel = KMeans(n_clusters=k, n_init=200)
    kmeanModel.fit(trainingData)
    inertias.append(kmeanModel.inertia_)
```




```
In [57]: from kneed import KneeLocator
         kn = KneeLocator(K, inertias, curve='convex', direction='decreasing')
         k=kn.knee
         k
```

```
Out[57]: 6
```

The elbow method gives 6 as an optimal k for k-means clustering, and we will run the k-means algorithm based on the proposed k:

```
In [48]: kmeanModel = KMeans(n_clusters=k)
         kmeanModel.fit(trainingData)
```

```
Out[48]: KMeans(n_clusters=6)
```

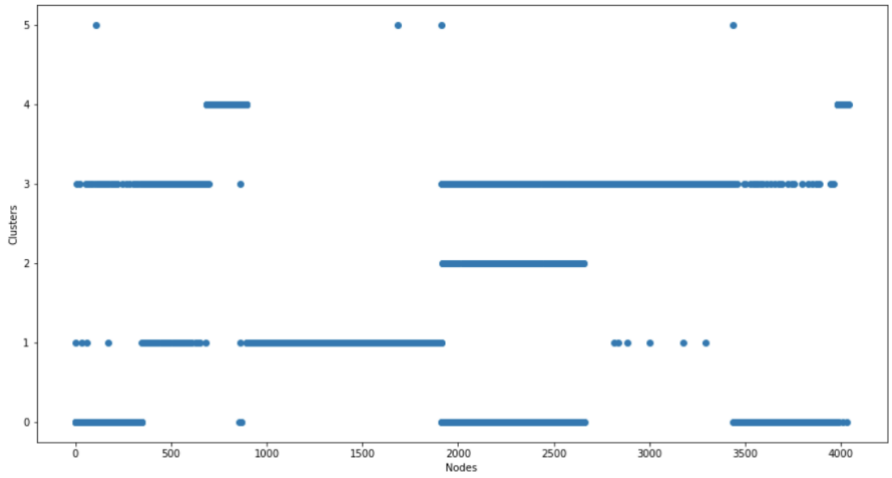
```
In [49]: Nodes = pd.DataFrame([i for i in G], columns = ['Node'])
         Clusters = pd.DataFrame(kmeanModel.labels_, columns = ['Cluster'])
         nodeByCluster = pd.concat([Nodes, Clusters], axis = 1)
         nodeByCluster
```

Out[49]:

	Node	Cluster
0	0	1
1	1	0
2	2	0
3	3	0
4	4	0
...	...	...
4034	4034	4
4035	4035	4
4036	4036	4
4037	4037	4
4038	4038	4

4039 rows × 2 columns

The scatterplot below shows the distribution of the nodes by clusters:



Visualization of different cluster on the graph, the nodes of the same cluster have the same color.

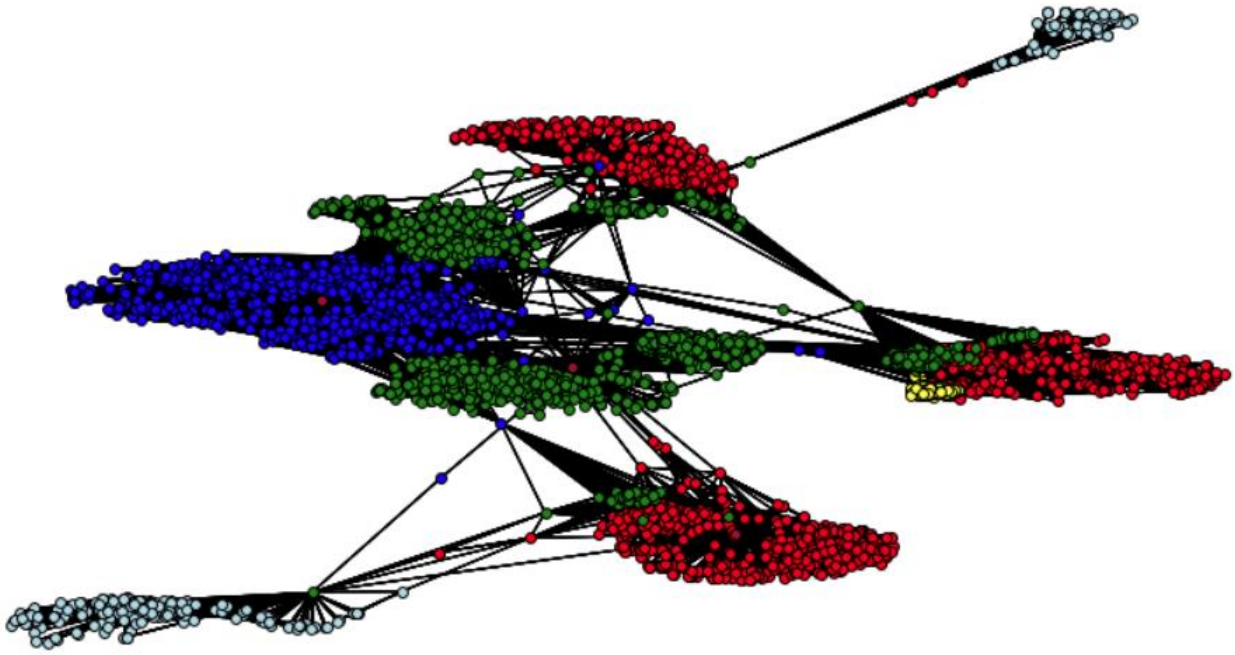


Figure 24: K-means (centrality measures) clustering with k= 6(by elbow method) of Facebook graph

ii. Zachary dataset

Constructing the network of Zachary dataset:

```
In [2]: G = nx.karate_club_graph()
        pos = nx.spring_layout(G)

In [3]: plt.figure(figsize=(20,10))
        nx.draw_networkx(G,pos=pos,with_labels=True)
```

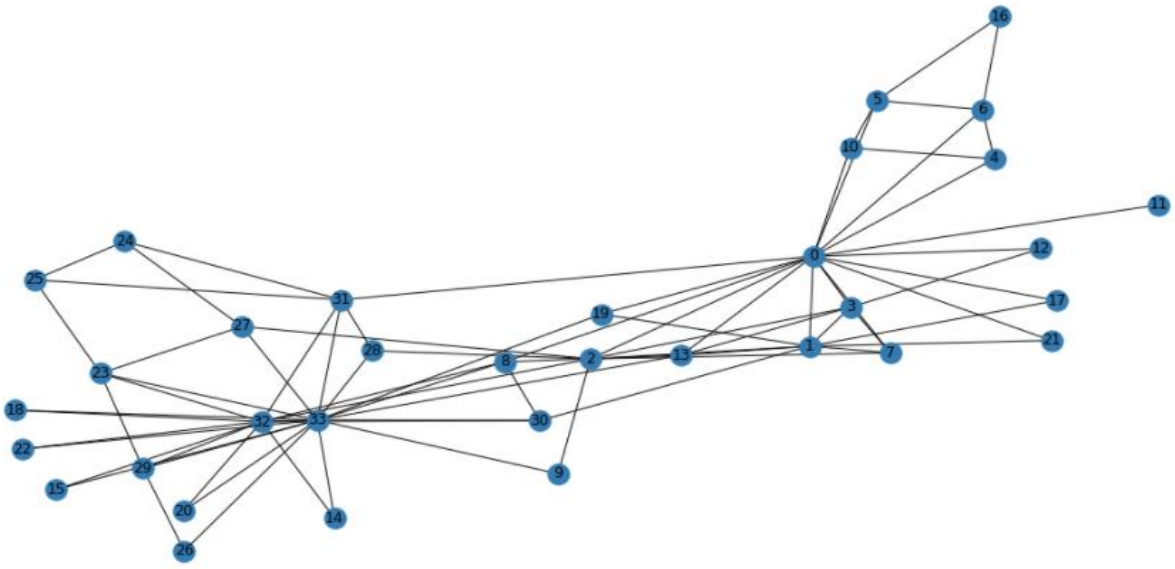


Figure 25: Zachary's karate club graph

And here it is the true labeling of this dataset:

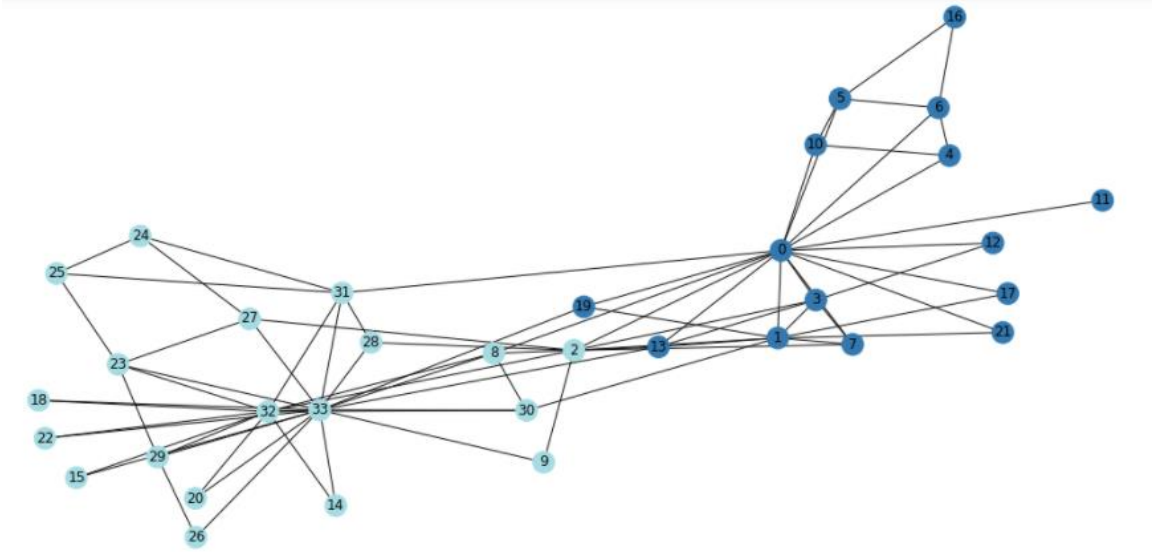
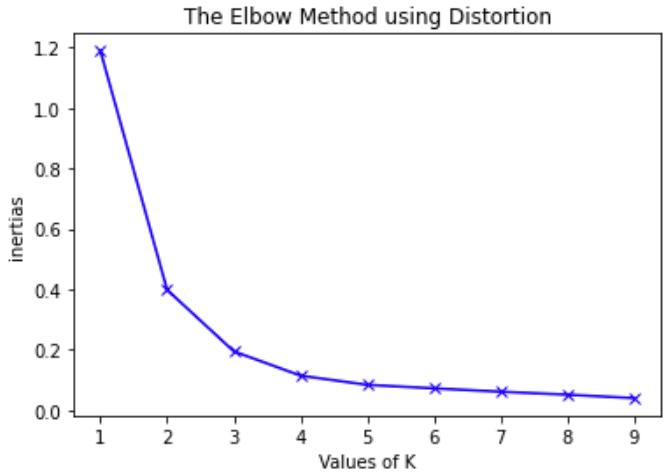


Figure 26: True labeling of Zachary graph

- **Centrality measures**

After applying the elbow method on Zachary dataset based on centrality measures, we got 3 as the optimal k value which is close enough to the real optimal value (2).




```
In [24]: from kneed import KneeLocator
         kn = KneeLocator(K, inertias, curve='convex', direction='decreasing')
         k=kn.knee
         k

Out[24]: 3
```

However, we're going to use k = 3 to cluster the network nodes.

This is a visualization of the output result:

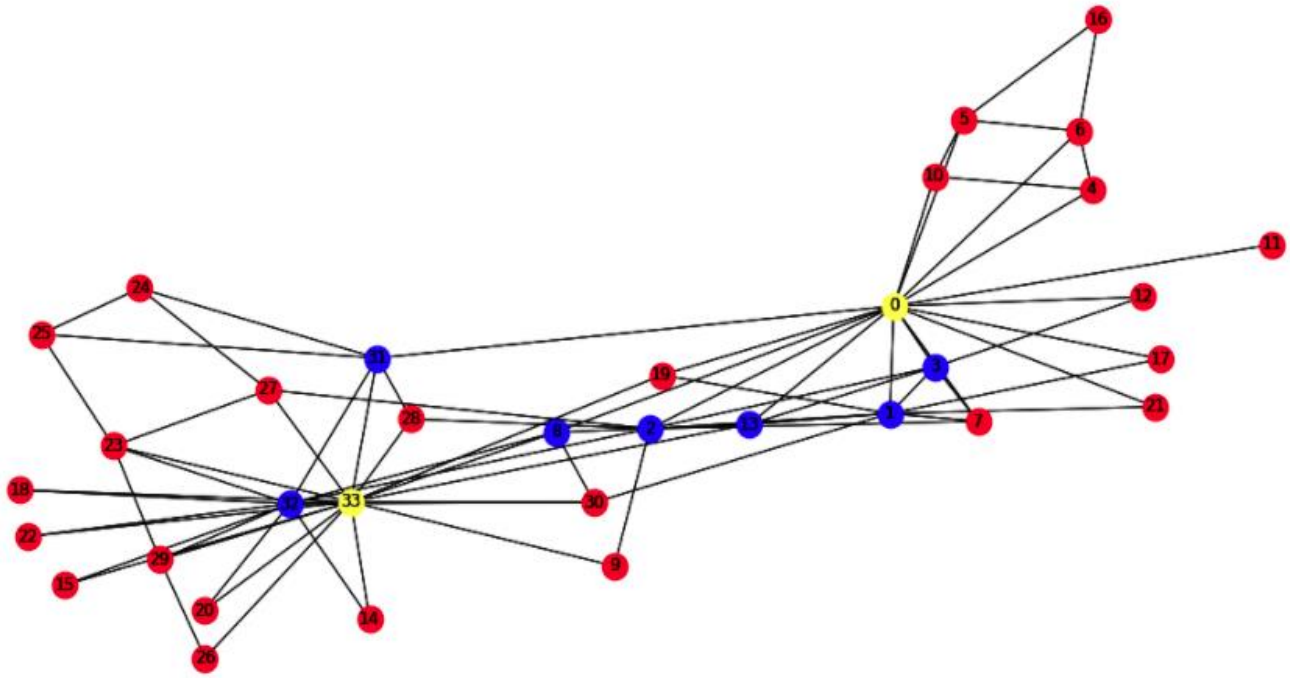


Figure 27: K-means (centrality measures) clustering with k=3(by elbow method) of Zachary graph

On the one hand the obtained clusters look totally different to the real clusters, which means the k-means using the centrality measures doesn't succeed to group nodes by their real communities.

On the other hand, k-means using centrality measures did exactly what we concluded in the Discuss section above, which we can notice in the graph above, 0 and 33 have the best influence in the network, so they are grouped in one cluster. The next 7 nodes that have more influence in the network are grouped in the same cluster. And the rest of the nodes are gathered in another cluster. Therefore, the clustering here is by the degree of influence.

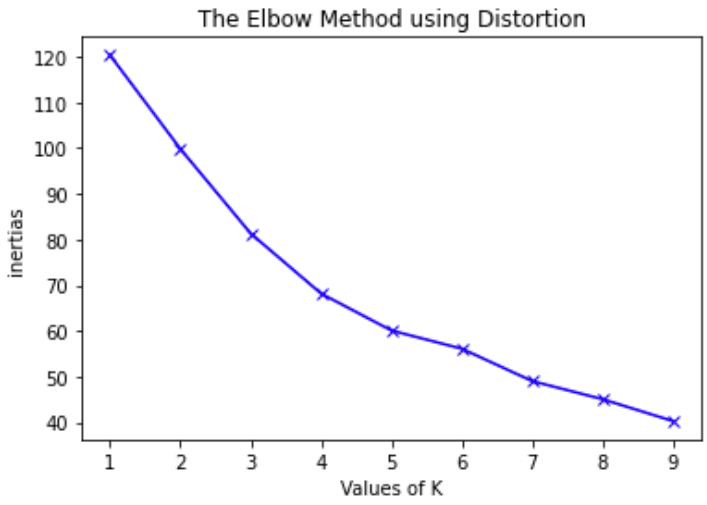
- **Adjacency matrix**

Constructing of the adjacency matrix of Zachary network:

```
In [5]: K =range(1, 10)
        inertias=[]
        X = nx.to_numpy_array(G, nodelist=[i for i in range(34)])
        X
```

```
Out[5]: array([[0., 1., 1., ..., 1., 0., 0.],
               [1., 0., 1., ..., 0., 0., 0.],
               [1., 1., 0., ..., 0., 1., 0.],
               ...,
               [1., 0., 0., ..., 0., 1., 1.],
               [0., 0., 1., ..., 1., 0., 1.],
               [0., 0., 0., ..., 1., 1., 0.]])
```

We did apply the elbow method on the same dataset again, but this time using its adjacency matrix and we got 4 this time:




```
In [150]: kn = KneeLocator(K, inertias, curve='convex', direction='decreasing')
          k=kn.knee
          k
```

```
Out[150]: 4
```

We applied k-means with k=4 and we got this:

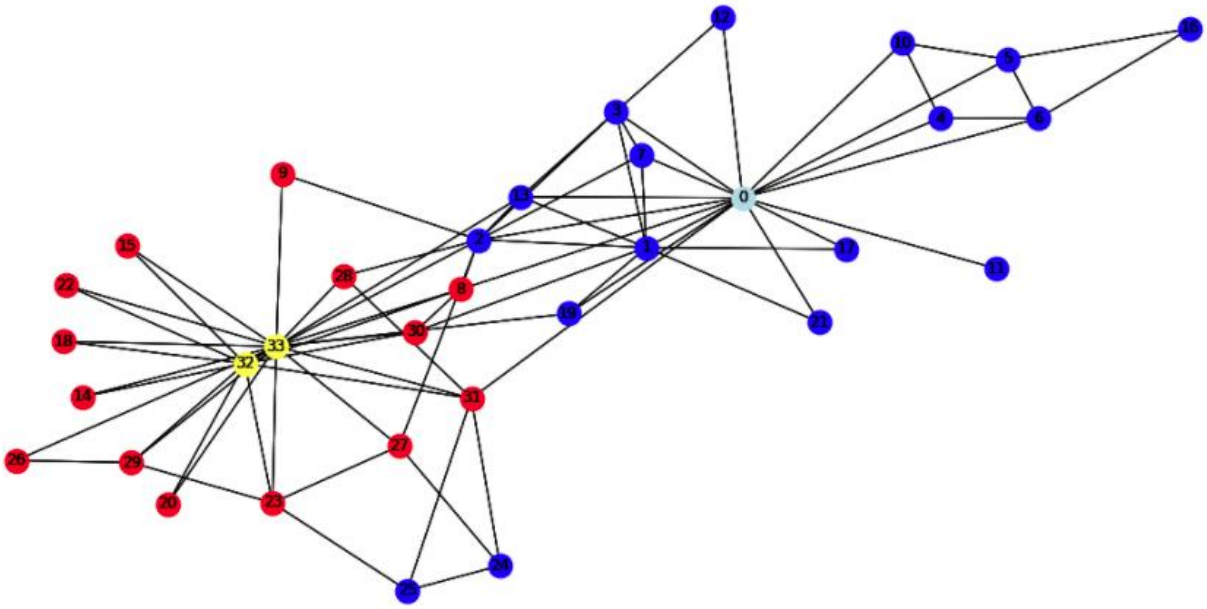


Figure 28: K-means (adjacency matrix) clustering with k=4(by elbow method) of Zachary graph

The obtained result looks close to the real labeling which proves the performance of using the adjacency matrix.

1.1.1. Silhouette method

The silhouette Method is also used to determine the ideal number of clusters, as well as to interpret and validate consistency within data clusters. The silhouette method calculates each point's silhouette coefficients, which measure how similar a point is to its own cluster compared to other clusters.

i. Facebook network dataset

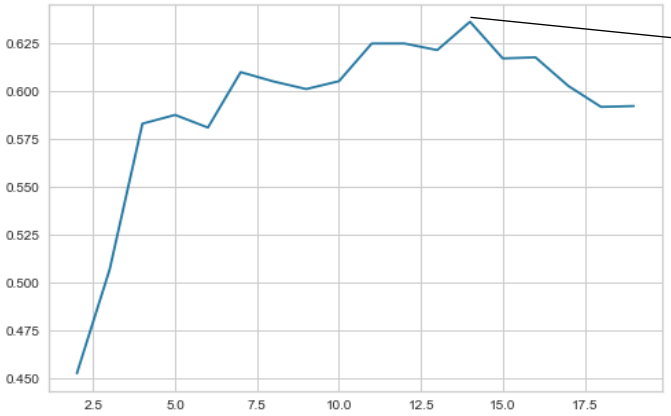
- Centrality measures

```
In [16]: s_score = []
for i in range(2,20):
    km = KMeans(n_clusters = i , n_init = 1000)
    km.fit_predict(trainingData)
    # Calculate Silhoutte Score
    score = silhouette_score(trainingData, km.labels_)
    s_score.append(score)

fig, ax = plt.subplots()

ax.plot([i for i in range(2,20)], s_score)
```

```
Out[16]: [<matplotlib.lines.Line2D at 0x283b463d760>]
```



We see that the silhouette score is maximized at k = 14. So, we will take 14 clusters

And here is the visualization of the output clustering:

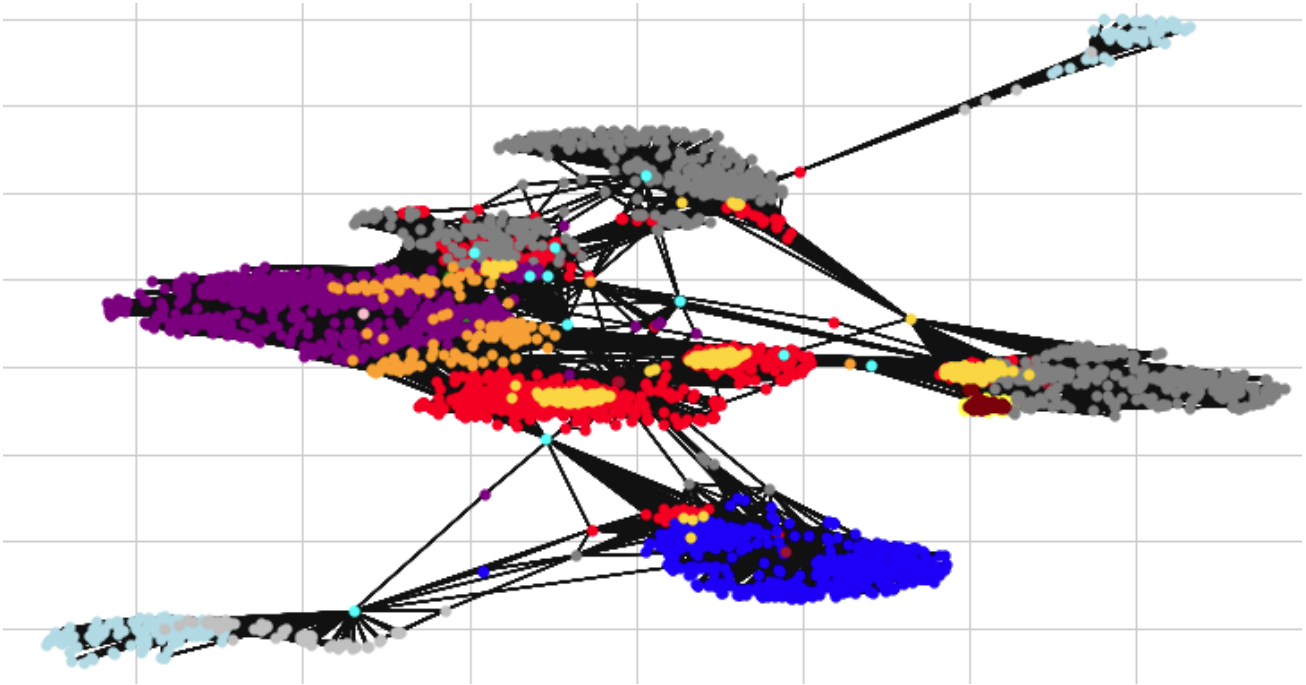


Figure 29: K-means (centrality measures) clustering with k=14(by silhouette method) of Facebook graph

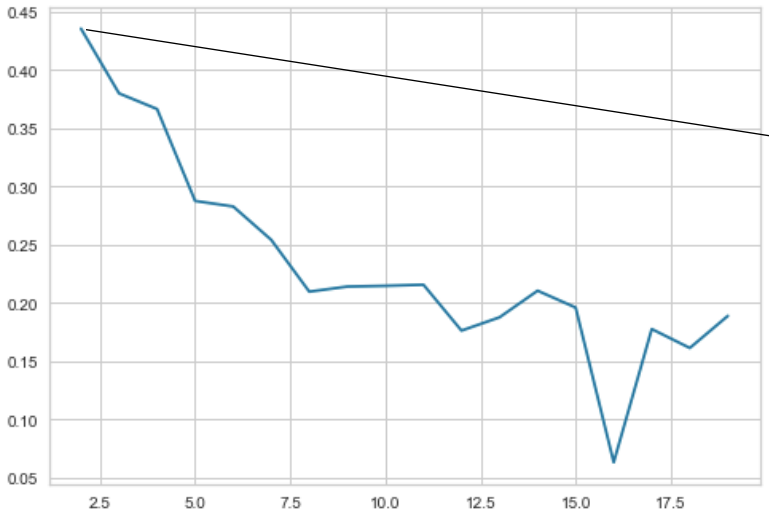
- **Adjacency matrix**

```
In [14]: X=nx.to_numpy_array(G, nodelist=[i for i in range(4039)])
         s_score = []
         for i in range(2,20):
             km = KMeans(n_clusters = i , n_init = 1000)
             km.fit_predict(X)
             # Calculate Silhoutte Score
             score = silhouette_score(X, km.labels_)
             s_score.append(score)

         fig, ax = plt.subplots()

         ax.plot([i for i in range(2,20)], s_score)

Out[14]: [<matplotlib.lines.Line2D at 0x283b4676370>]
```



We see that the silhouette score is maximized at k = 2. So, we will take 2 clusters

And here is the visualization of the output clustering using 2 clusters:

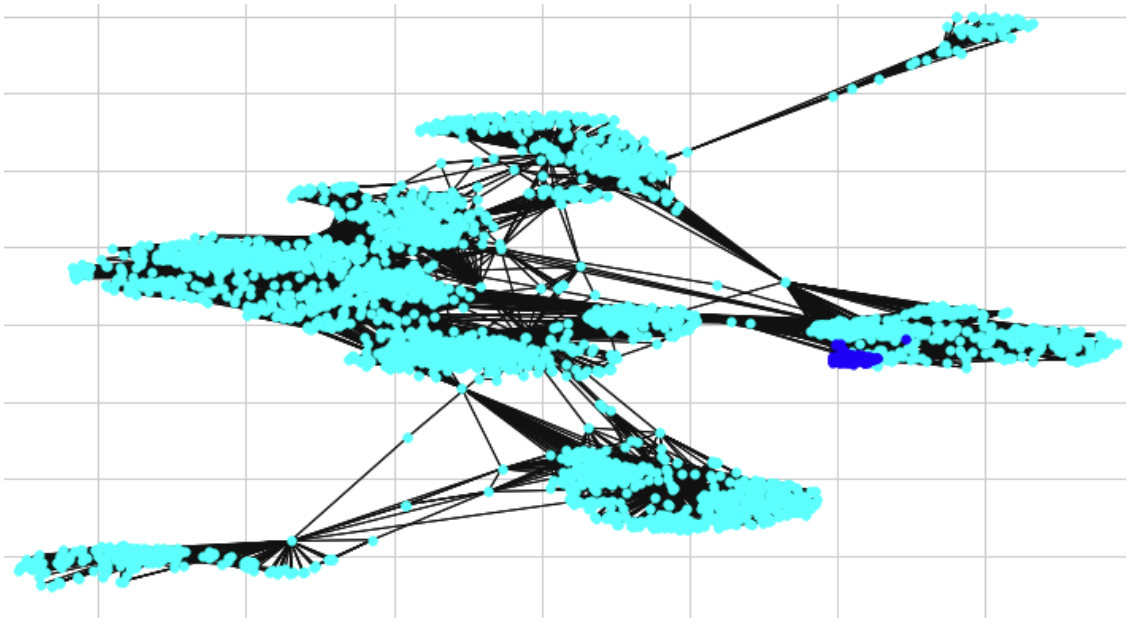


Figure 30: K-means (adjacency matrix) clustering with k=2(by silhouette method) of Facebook graph

ii. Zachary dataset

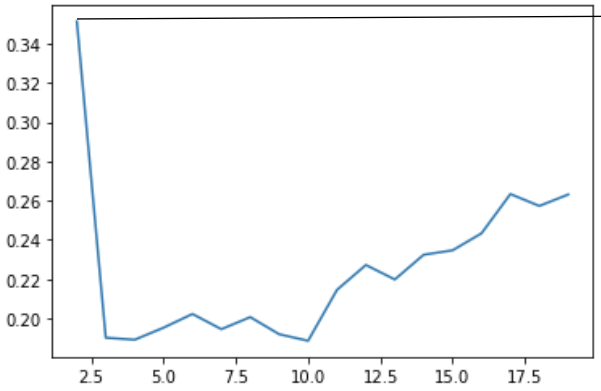
We're going to use here the adjacency matrix only because of the bad performance gotten using the centrality measures before.

```
In [7]: s_score = []
        for i in range(2,20):
            km = KMeans(n_clusters = i , n_init = 1000)
            km.fit_predict(A)
            # Calculate Silhoutte Score
            score = silhouette_score(A, km.labels_)
            s_score.append(score)
```

```
In [8]: fig, ax = plt.subplots()

        ax.plot([i for i in range(2,20)], s_score)
```

```
Out[8]: [<matplotlib.lines.Line2D at 0x23ae1d2cb20>]
```



We see that the silhouette score is maximized at k = 2. So, we will take 2 clusters

It seems like the silhouette method gives the best result in term of finding the optimal k value. And after applying the k-means with this k value we got the perfect clustering.

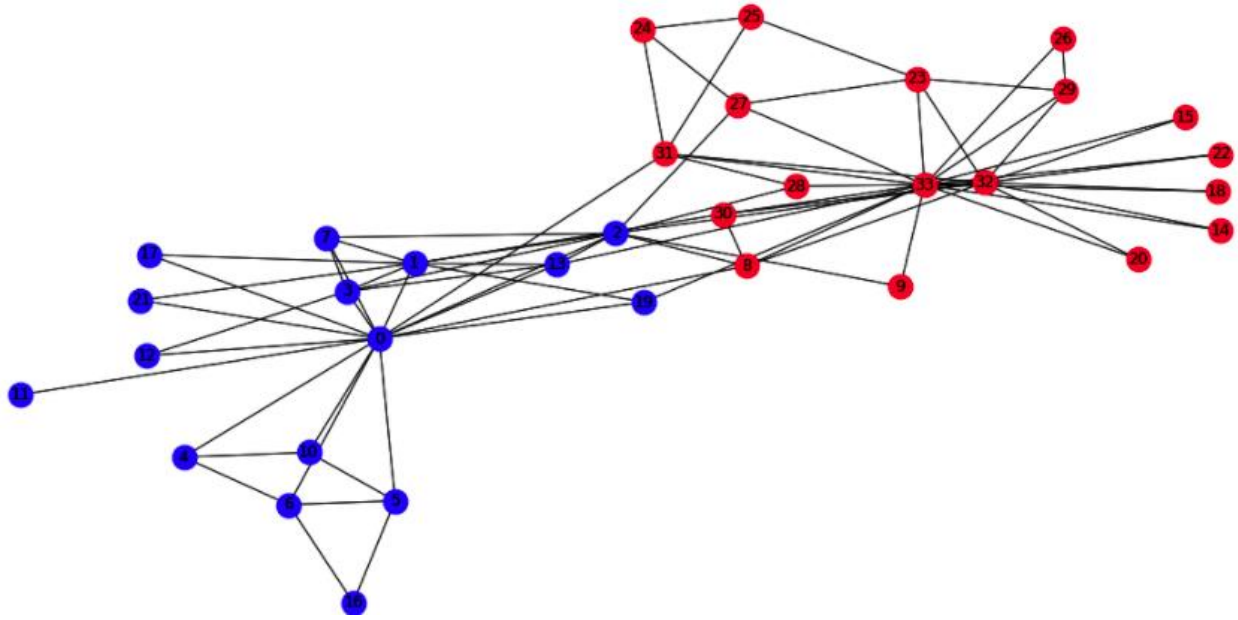


Figure 31: K-means (adjacency matrix) clustering with k=2(by silhouette method) of Zachary graph

Conclusion

Community detection is one of the most important issues in social network analysis. K-means algorithm has been shown its efficiency and simplicity in clustering large scale dataset into smaller and simple datasets. However, k-means has shown also its sensitivity to the initial centroids, choosing different initial values leads to different result. To solve this issue, in this report we propose to choose the top-k influential nodes as the initial centroids, these nodes are identified using a proposed method (TOPSIS).

Identifying the top influential nodes has been also a big challenge. The regular centrality measures have shown some limitation. Therefore, we used the TOPSIS instead. Later, we the SI model has been used to evaluate this method. Thus, the TOPSIS method performed better than DC, CC and EC and as much as BC.

Two types of datasets of network have been used, the network's adjacency matrix and centrality measures of each one of its nodes. Thus, after some experimental analysis, we have concluded that the adjacency matrix has performed better than the centrality measures in term of community detection using k-means algorithm. In fact, working with centrality measures led to a clustering based on the influence of each node as explained before in the report.

In the end, we tried to handle one k-means problems which is choosing the optimal k value. To do this, we attempted two well-known methods, elbow and silhouette methods on two different datasets.

References

[1] P. Joshi, "Getting Started with Community Detection in Graphs and Networks," 7 2 2022. [Online]. Available: https://www.analyticsvidhya.com/blog/2020/04/community-detection-graphs-networks/.

[2] "networkx.algorithms.centrality.eigenvector_centrality," 8 2 2022. [Online]. Available: https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.centrality.eigenvector_centrality.html#rc5278aad1436-2.

[3] R. Rahim, "TOPSIS Method Application for Decision Support," 8 2 2022. [Online]. Available: https://iopscience.iop.org/article/10.1088/1742-6596/1028/1/012052/pdf.

[4] "SI," 9 2 2022. [Online]. Available: https://ndlib.readthedocs.io/en/latest/reference/models/epidemics/SIm.html#id2.

[5] A. Trevino, "Introduction to K-means Clustering," 9 2 2022. [Online]. Available: https://blogs.oracle.com/ai-and-datascience/post/introduction-to-k-means-clustering.

[6] "An Improved Louvain Algorithm for Community Detection," hindawi, [Online]. Available: https://www.hindawi.com/journals/mpe/2021/1485592/. [Accessed 20 2 2022].

[7] "Markov clustering," DAVE TANG'S BLOG, [Online]. Available: https://davetang.org/muse/2014/01/23/markov-clustering/. [Accessed 20 2 2022].

[8] "Silhouette Algorithm to determine the optimal value of k," GeeksforGeeks, 6 6 2019. [Online]. Available: https://www.geeksforgeeks.org/silhouette-algorithm-to-determine-the-optimal-value-of-k/. [Accessed 20 2 2022].

[9] "What is Python? Executive Summary," 10 2 2022. [Online]. Available: https://www.python.org/doc/essays/blurb/.

[10] "The Python Math Library," 10 2 2022. [Online]. Available: https://stackabuse.com/the-python-math-library/.

[11] "What is NumPy?," 10 2 2022. [Online]. Available: https://numpy.org/doc/stable/user/whatisnumpy.html.

[12] "Pandas," 10 2 2022. [Online]. Available: https://mode.com/python-tutorial/libraries/pandas/.

[13] "NetworkX Package – Python Graph Library," 10 2 2022. [Online]. Available: https://www.askpython.com/python-modules/networkx-package.

[14] "NDlib - Network Diffusion Library," 10 2 2022. [Online]. Available: https://ndlib.readthedocs.io/en/latest/.

[15] "CDlib - Community Discovery Library," 10 2 2022. [Online]. Available: https://cdlib.readthedocs.io/en/latest/.

[16] "Scikit Learn Tutorial," tutorialspoint, [Online]. Available: https://www.tutorialspoint.com/scikit_learn/index.htm. [Accessed 20 2 2022].

[17] "Everything You Need To Know About Jupyter Notebooks," 10 2 2022. [Online]. Available: https://towardsdatascience.com/everything-you-need-to-know-about-jupyter-notebooks-10770719952b.

[18] "Network Graph: What It Is, When To Use It, And The DataWalk Solution," 2 7 2022. [Online]. Available: https://datawalk.com/glossary/network-graph/.