Handout

Introduction to Social Networks using NetworkX in Python

Ever wondered how the most popular social networking site Facebook works? How we are connected with friends using just Facebook? So, Facebook and other social networking sites work on a methodology called social networks. Social networking is used in mostly all social media sites such as Facebook, Instagram, and LinkedIn, etc. It has a significant effect on marketers to engage customers. Social networks use graphs for creating a network. Their nodes are people and edges are their connection between each other. Two nodes with edges connected are friends. Now let's see an example for understanding what is social networks.

The network of 50 students in a class

The network of 50 people

The most important python library used in social networking is **Networkx.**

NetworkX

NetworkX is a graph package that is used to create and modify different types of graphs. It provides a rapid development environment for collaborative, multidisciplinary projects.

Installation:

```
pip install networkx
```

After starting python, we have to import networkx module:

```
import networkx as nx
```

Basic inbuilt graph types are:

- **Graph:** This type of graph stores nodes and edges and edges are un-directed. It can have self-loops but cannot have parallel edges.

- **Di-Graph:** This type of graph is the base class for directed graphs. It can have nodes and edges and edges are directed in nature. It can have self-loops but parallel edges are not allowed in Di-Graph.
- **Multi-Graph:** This type of graph is an undirected graph class that can store multi or parallel edges. It can have self-loops as well. Multi-edges are multiple edges between 2 nodes.
- **Multi-DiGraph:** This type of graph is a directed graph class that can store multi edges. It can have self-loops as well. Multi-edges are multiple edges between 2 nodes.

Example of Graph creation :

```
# import networkx library

import networkx as nx


# create an empty undirected graph

G = nx.Graph()


# adding edge in graph G

G.add_edge(1, 2)

G.add_edge(2, 3, weight=0.9)
```

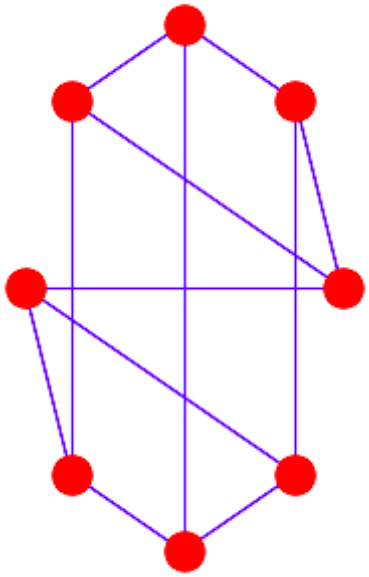
Drawing of graph:

Drawing can be done using Matplotlib.pyplot library.

```
# import matplotlib.pyplot library

import matplotlib.pyplot as plt


# import networkx library

import networkx as nx


# create a cubical empty graph

G = nx.cubical_graph()


# plotting the graph

plt.subplot(122)
```

```
# draw a graph with red

# node and value edge color

nx.draw(G, pos = nx.circular_layout(G),

                    node_color = 'r',

                    edge_color = 'b')
```

Output:



Graph Edge Removal:

To remove an edge from the graph, use the **remove_edge()** method of graph object.

Syntax: *G.remove_edge(u, v)*

Parameters:

- ***u:*** *first node*
- ***v:*** *second node*

Return*: None*

Graph Node Removal:

To remove a node from the graph, use the **remove_node()** method of graph object.

Syntax: *G.remove_node(u)*

Parameter: *Node to remove*

Return: *None*

Show the Adjacent vertices:

```
# import networkx library
```

```
import netwokx as nx


# create an empty undirected graph
G = nx.Graph()


# add edge to the graph
G.add_edge('1', '2')
G.add_edge('2', '3')


# print the adjacent vertices
print(G.adj)
```

Output:

```
{'1': {'2': {}}, '2': {'1': {}, '3': {}}, '3': {'2': {}}}
```

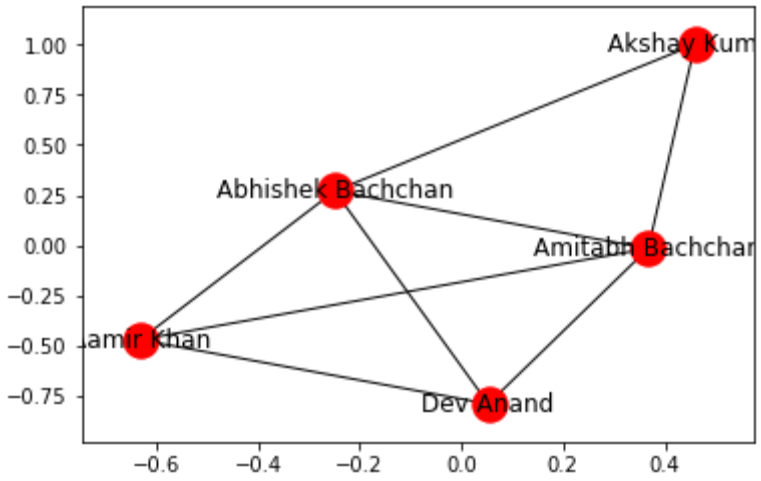
Symmetric Networks

The first network of actors that we created above is a symmetric network because the relationship "working together in a movie" is a symmetric relationship. If A is related to B, B is also related to A. Let us create the network we saw above in NetworkX.

We will be using the `Graph()` method to create a new network and `add_edge()` to add an edge between two nodes.

```
import networkx as nx

G_symmetric = nx.Graph()

G_symmetric.add_edge('Amitabh Bachchan','Abhishek Bachchan')

G_symmetric.add_edge('Amitabh Bachchan','Aamir Khan')

G_symmetric.add_edge('Amitabh Bachchan','Akshay Kumar')

G_symmetric.add_edge('Amitabh Bachchan','Dev Anand')

G_symmetric.add_edge('Abhishek Bachchan','Aamir Khan')

G_symmetric.add_edge('Abhishek Bachchan','Akshay Kumar')

G_symmetric.add_edge('Abhishek Bachchan','Dev Anand')

G_symmetric.add_edge('Dev Anand','Aamir Khan')
```

Let us now visualize the network we have just constructed using `nx.draw_networkx(G_symmetric)`.



Asymmetric Networks

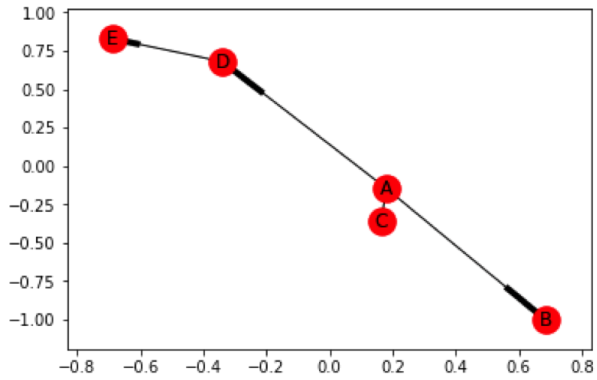
What if the relationship between nodes is 'child of', then the relationship is no longer symmetric. If A is the child of B, then B is not a child of A. Such a network where the relationship is asymmetric (A is related to B, does not necessarily means that B is associated with A) is called an Asymmetric network. We can build the asymmetric network in NetworkX using DiGraph method, which is short of Directional Graph. Let us make an asymmetric graph.

```
G_asymmetric = nx.DiGraph()

G_asymmetric.add_edge('A','B')

G_asymmetric.add_edge('A','D')

G_asymmetric.add_edge('C','A')

G_asymmetric.add_edge('D','E')
```

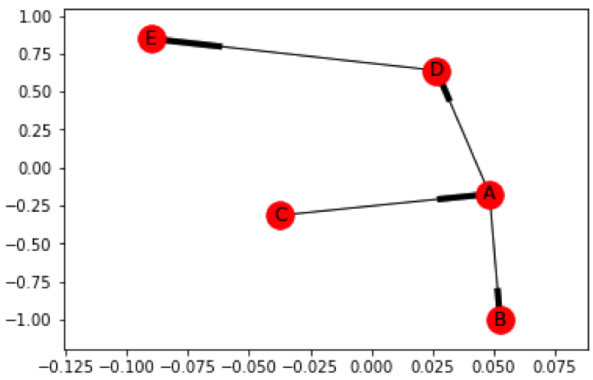
Now we visualize it. We can use the draw_networkx() function as before. However, it is possible that nodes do not separate out and are distinctly visible in the network drawn. To take care of this, we can use the function to force a layout, which positions the nodes in a manner that we can distinctly see them. We can accomplish this using spring_layout() function, followed by the draw_networkx() function.

```
nx.spring_layout(G_asymmetric)

nx.draw_networkx(G_asymmetric)
```

Below you can see the network with and without using the layout command. The one made using the layout command has more clarity.



Without using layout command

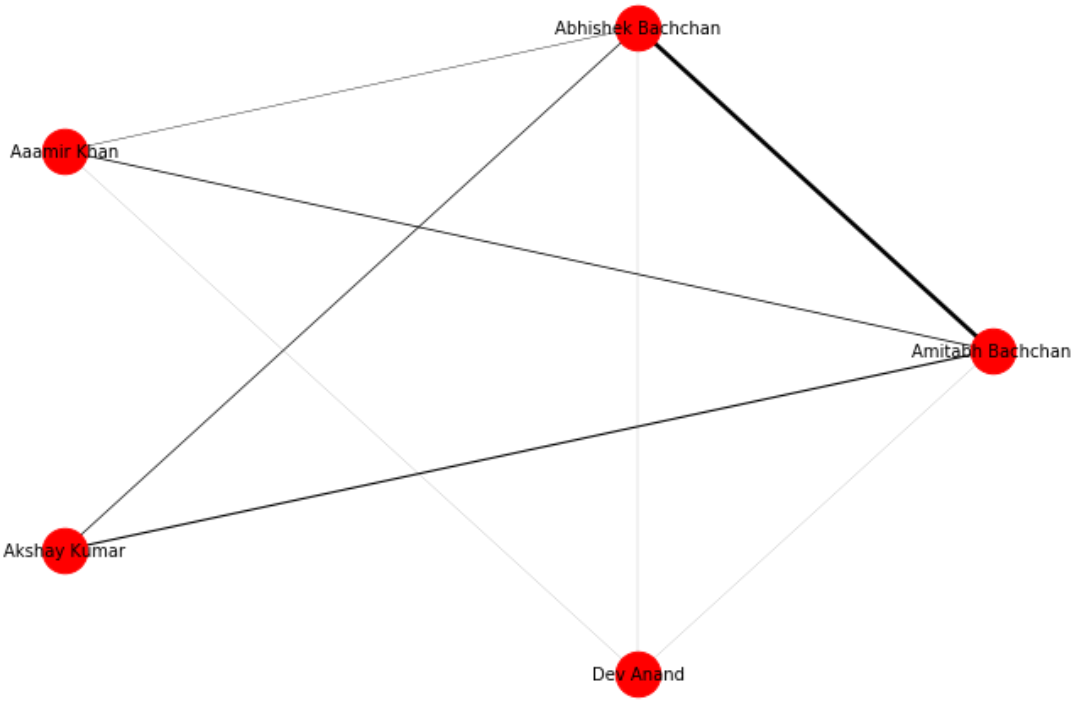


Using the layout command

Weighted Networks

Till now we had networks without weights, but it is possible that networks are made with weights, for example, if in our initial network we consider the number of movies done together as a weight, we will get a Weighted Network. Let us make one again of the actors, but this time we add weight to the network, each edge has a weight signifying the number of movies they have done together.

```
G_weighted = nx.Graph()
G_weighted.add_edge('Amitabh Bachchan','Abhishek Bachchan', weight=25)
G_weighted.add_edge('Amitabh Bachchan','Aaamir Khan', weight=8)
G_weighted.add_edge('Amitabh Bachchan','Akshay Kumar', weight=11)
G_weighted.add_edge('Amitabh Bachchan','Dev Anand', weight=1)
G_weighted.add_edge('Abhishek Bachchan','Aaamir Khan', weight=4)
G_weighted.add_edge('Abhishek Bachchan','Akshay Kumar',weight=7)
G_weighted.add_edge('Abhishek Bachchan','Dev Anand', weight=1)
G_weighted.add_edge('Dev Anand','Aaamir Khan',weight=1)
```



The figure above shows the weighted network of actors in a circular layout. The edge width specifies the weight between two nodes.

Multigraph

We can give different attributes to the edges. For example, we can define a relation of neighbor between two nodes 'A' and 'B' using relation attribute. If within a network two nodes are connected with two different edges (relations) we have a multigraph. We can make a multigraph utilizing the MultiGraph class.

```
G = nx.MultiGraph()
G.add_edge('A','B',relation ='neighbor')
G.add_edge('A','B',relation='friend)
G.add_edge('B','C', relation='neighbor')
G.add_edge('D','C',relation='friend')
```

his code will construct a graph with two edges between A and B. We can check the connections using G.edges(), the output would show:

```
MultiEdgeDataView([('A', 'B', {'relation': 'neighbor'}),
('A', 'B', {'relation': 'friend'}), ('B', 'C', {'relation':
'neighbor'}), ('B', 'D', {'relation': 'neighbor'}), ('C',
'D', {'relation': 'friend'})])
```

Network Connectivity

Now the network is made, can we know more about a particular node in the network? Well yes, let us explore some of them.

Degree

Degree of a node defines the number of connections a node has. NetworkX has the function `degree` which we can use to determine the degree of a node in the network.

```
nx.degree(G_symmetric, 'Dev Anand`)
```

This will return a value of 3, as Dev Anand has worked with only three actors in the network.

Clustering Coefficient

It is observed that people who share connections in a social network tend to form associations. In other words, there is a tendency in a social network to form clusters. We can determine the clusters of a node, **Local Clustering Coefficient**, which is the fraction of pairs of the node's friends (that is connections) that are connected with each other. To determine the local clustering coefficient, we make use of `nx.clustering(Graph, Node)` function.

In the symmetric Actor-network, you will find that ***Dev Anand*** has a local clustering coefficient of 1 and ***Abhishek Bachchan*** has a local clustering coefficient of 0.67.

The average clustering coefficient (sum of all the local clustering coefficients divided by the number of nodes) for the symmetric Actor-network is 0.867. We can obtain it using:

```
nx.average_clustering(G_symmetric)
```

Distance

We can also determine the shortest path between two nodes and its length in NetworkX using `nx.shortest_path(Graph, Node1, Node2)` and `nx.shortest_path_length(Graph, Node1, Node2)` functions respectively.

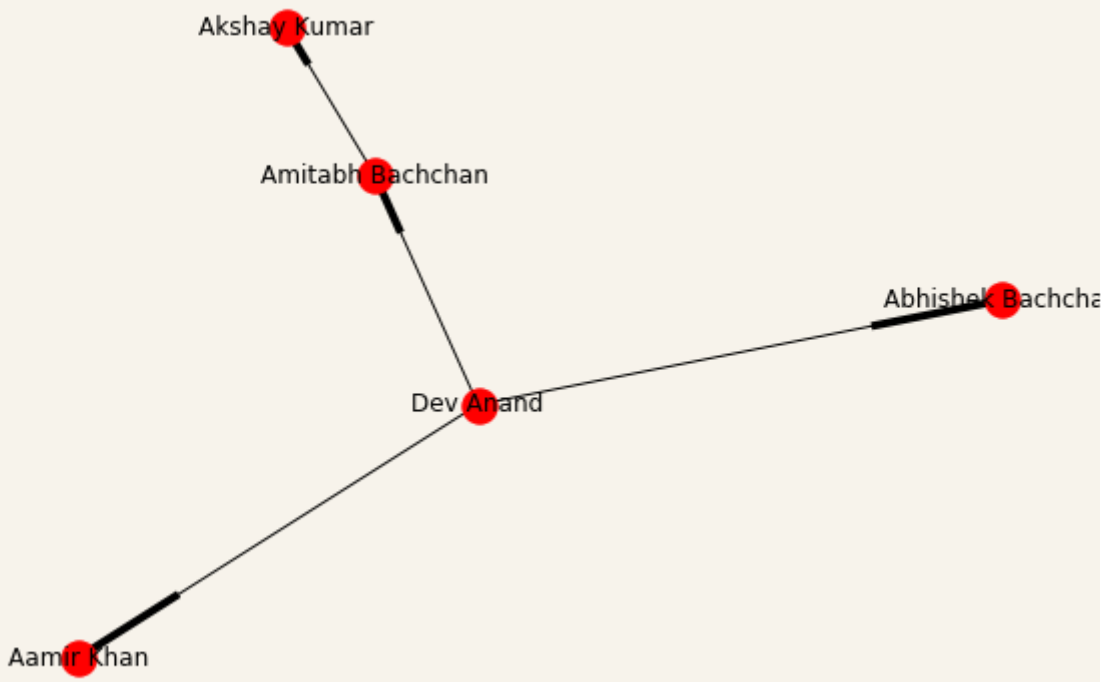
Executing

```
nx.shortest_path(G_symmetric, 'Dev Anand', 'Akshay Kumar')
```

Returns

```
['Dev Anand', 'Amitabh Bachchan', 'Akshay Kumar']
```

We can find the distance of a node from every other node in the network using breadth-first search algorithm, starting from that node. networkX provides the function `bfs_tree` to do it. And so if you try `T = nx.bfs_tree(G_symmetric, 'Dev Anand')` and now draw this tree, we will get a network structure telling how we can reach other nodes of the network starting from ***Dev Anand***



Eccentricity

Eccentricity of a node A is defined as the largest distance between A and all other nodes. It can be found using `nx.eccentricity()` function. In the symmetric Actor-network, ***Dev Anand*** has an eccentricity of 2, and ***Abhishek Bachchan*** has an eccentricity of 1 (It is connected to all).