Future link prediction for Facebook entities

A Social and Information Networks(CSE3021) project report

Submitted to:

Dr. Punitha.K

Submitted by:

Alpanshu Kataria (18BCE1267)

Mehul Malik (18BCE1097)

In partial fulfilment for the award of the degree of

Bachelor of Technology

in

Computer Science and Engineering

July-November 2020

TABLE OF CONTENTS

Page

1.Abstract:

Most social media platforms, including Facebook, can be structured as graphs. The registered users are interconnected in a universe of networks. And to work on these networks and graphs, we need a different set of approaches, tools, and algorithms (instead of traditional machine learning methods). So in this project, we will solve a social network problem with the help of graphs and machine learning. We will understand the core concepts and components of link prediction and take up a Facebook case study and implement it in Python. Given an instance of set of nodes (users) in a social network graph, the aim is to find the influencing (important) users and to predict the likelihood of a future association (edge) between two nodes, knowing that there is no association between the nodes in the current state of the graph.

2.Introduction:

Starting from the beginning of this century, social networks have significantly evolved, and the progression of social media platforms (Facebook, Twitter, LinkedIn, etc.) has been well documented in a significant number of publications. In the contemporary era of connectivity, the majority of organizations complement their traditional marketing strategies with digital campaigns that rely heavily on social media channels. Due to its increase in popularity, social media has become integral to organizations' advertising and marketing campaigns, representing a cost-effective and efficient channel through which companies target and communicate with members of a given audience. Since their inception in the late 90s, the value of social networks (SNs) has dramatically increased to reach billions of dollars. As such, they represent an attractive investment proposition, especially in the marketing sector. Their rise in popularity and social and economic implications has also attracted significant research attention. One area of interest particularly notable in recent times is the prediction of missing links

Let's define a social network first before we dive into the concept of link prediction: A social network is essentially a representation of the relationship between social entities such as people, organisations, governments, political parties etc. The interactions among these entities generate unimaginable amounts of data in the form of posts, chat messages, tweets, likes, comments, shares, etc. This opens up a window of opportunities and use cases we can work on. That brings us to Social Network Analytics (SNA). We can define it as a combination of several activities that are performed on social media. These activities include data collection from online social media sites and using that data to make business decisions. One of the ways of using that data is by link prediction. Link prediction is one of the most important research topics in the field of graphs and networks**.** The objective of link prediction is to identify pairs of nodes that will either form a link or not in the future.

3.Feasibility study:

Link prediction techniques have found a large number of applications in very different fields. Any domain where entities interact in a structured way can potentially benefit from link prediction.

The edges described in the problem statement could be of any form: friendship, collaboration, following or mutual interests. Here, we specifically study and build our model over Facebook's social network, with the following areas of motivation:

- General application of friends recommendation to a particular user.
- Predicting hidden links in a social network group formed by terrorists along with identification of their leaders/ key influencers.
- Targeted marketing of products: Marketing through highly influential individuals and also identifying plausible customers.
- Suggesting promising interactions or collaborations that have not yet been identified within an organization.
- In Bioinformatics, link prediction can be used to find interactions between proteins.

The following model can be extended or modified to cater to the needs of various other social networks like Twitter, Google+, Foursquare, etc.

The benefits of social network analytics can be highly rewarding. Here are a few key benefits:

- Helps you understand your audience better
- Used for customer segmentation
- Used to design Recommendation Systems
- Detect fake news, among other things

Link prediction has a ton of use in real-world applications too. Here are some of the important use cases of link prediction:

- Predict which customers are likely to buy what products on online marketplaces like Amazon. It can help in making better product recommendations
- Suggest interactions or collaborations between employees in an organization
- Extract vital insights from terrorist networks

There are some problems as well which need to be addressed. Very few techniques adapt to the global structure of the network and no technique adapts to the local structure of networks. The main difficulty when dealing with complex networks in practice is their size, which limits the kinds of techniques that can be applied. A temporal network is a network that evolves over time. For predicting the emergence of links in such a type of a network, time factors are crucial. The main problems and challenges of predicting links in a temporal network that require addressing are related to the changing nature of the network over time. Many methods and techniques have been proposed to address this problem. For example, there’s a technique to exploit the general network structure and to successfully predict links in the process. There’s a technique using graph theory to predict links, and used community structure information to do so. An efficient method proposed by produced far better results than many contemporary temporal link-prediction methods

4.Design and Flow of modules:

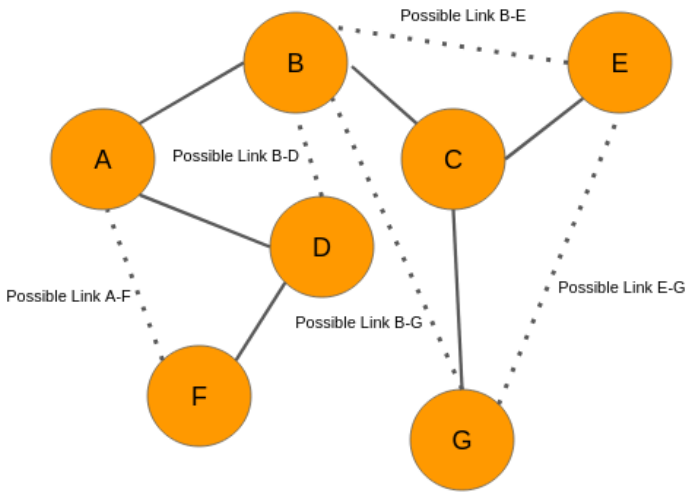
Following are the modules in which we have divided are project:

- Forming the strategy for making prediction model and setting the definite approach.
- Analysing the data
- Visualizing the data using various types of graphs
- Creating a train and test dataset from the available data and making a feature extraction model.
- Implementing different ML models and algorithms and doing a comparative analysis to find out where the prediction is best.

4.1 Strategy:

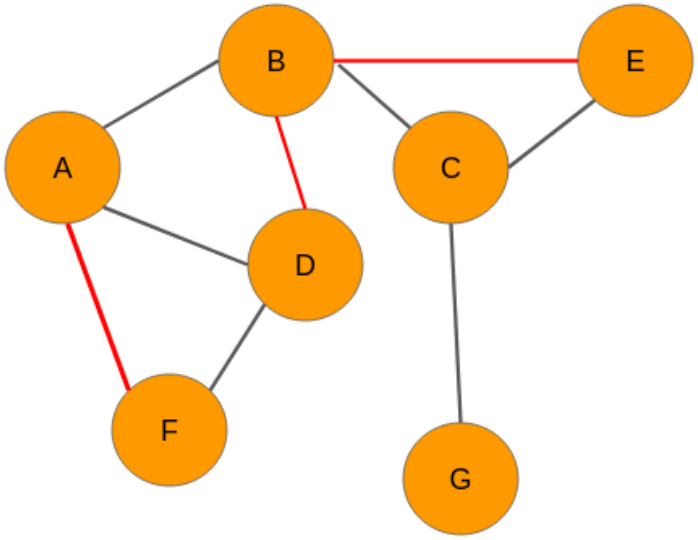
If we can somehow represent a graph in the form of a structured dataset having a set of features, then maybe we can use machine learning to predict the formation of links between the unconnected node-pairs of the graph.

Let's take a dummy graph to understand this idea. Given below is a 7 node graph and the unconnected node-pairs are AF, BD, BE, BG, and EG:



Graph at time t

Now, let's say we analyze the data and came up with the below graph. A few new connections have been formed (links in red):



Graph at time t+n

We need to have a set of predictor variables and a target variable to build any kind of machine learning model, right? So where are these variables? Well, we can get it from the graph itself! Let's see how it is done.

Our objective is to predict whether there would be a link between any 2 unconnected nodes. From the network at time t, we can extract the following node pairs which have no links between them:

1. A-F
2. B-D
3. B-E
4. B-G
5. E-G

Please note that, for convenience, I have considered only those nodes that are a couple of links apart.

The next step for us is to create features for each and every pair of nodes. The good news is that there are several techniques to extract features from the nodes in a network. Let's say we use one of those techniques and build features for each of these pairs. However, we still don't know what the target variable is. Nothing to worry about – we can easily obtain that as well.

Look at the graph at time *t+n*. We can see that there are three new links in the network for the pairs A-F, B-D, and BE respectively. Therefore, we will assign each one of them a value of 1. The node pairs B-G and E-G will be assigned 0 because there are still no links between the nodes.

Hence, the data will look like this:

Features	Link (Target Variable)
Features of A-F pair	1
Features of B-D pair	1
Features of B-E pair	1
Features of B-G pair	0
Features of E-G pair	0

Now that we have the target variable, we can build a machine learning model using this data to perform link prediction.

So, this is how we need to use social graphs at two different instances of time to extract the target variable, i.e., the presence of a link between a node pair. Keep in mind, however, that in real-world scenarios, we will have data of the present time only.

Extract data from a Graph for Building your Model

In the section above, we were able to get labels for the target variable because we had access to the graph at time t+n. However, in real-world scenarios, we would have just one graph dataset in hand. That's it!

Let's say we have the below graph of a social network where the nodes are the users and the edges represent some kind of relationship:

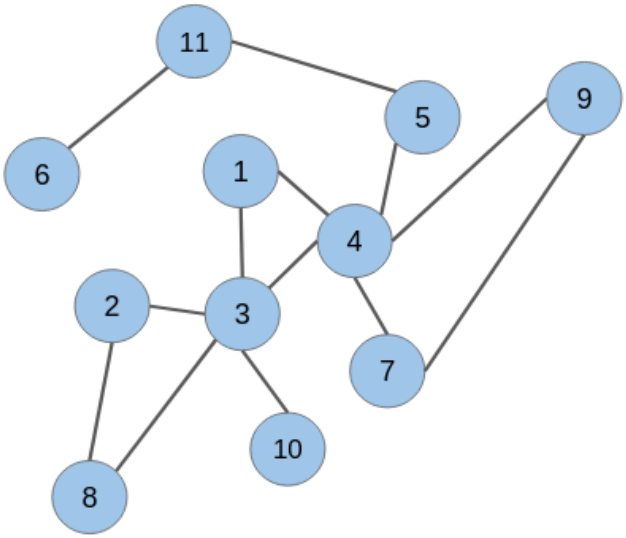


Fig. 1

The candidate node pairs, which may form a link at a future time, are **(1 & 2)**, **(2 & 4)**, **(5 & 6)**, **(8 & 10)**, and so on. We have to build a model that will predict if there would be a link between these node pairs or not. This is what link prediction is all about!

However, to build a link prediction model, we need to prepare a training dataset out of this graph. It can be done using a simple trick.

Picture this – how would this graph have looked like at some point in the past? There would be fewer edges between the nodes because connections in a social network are built gradually over time.

Hence, keeping this in mind, we can randomly hide some of the edges from the given graph and then follow the same technique as explained in the previous section to create the training dataset

Striking of the links from the graph:

While removing links or edges, we should avoid removing any edge that may produce an isolated node (node without any edge) or an isolated network. Let's take off some of the edges from our network:

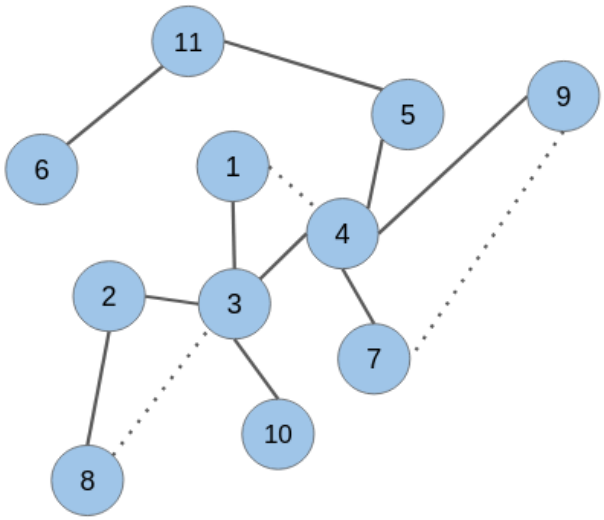


Fig. 2

As you can see, the edges in the node pairs (1 & 4), (7 & 9), and (3 & 8) have been removed.

Add labels to the extracted data:

Next, we would need to create features for all the unconnected node pairs including the ones for which we have omitted the edges. The removed edges will be labeled as '1' and the unconnected node pairs as '0'

It turns out that the target variable is highly imbalanced. This is what you will encounter in real-world graphs as well. The number of unconnected node pairs would be huge.Now we'll apply all of the above into a real-world scenario:

5.Risk Analysis:

Due to the vast amount of data in social networks, achieving efficiency within link-prediction methods represents a significant challenge. To address this problem, proposed two algorithms for link prediction. The proposed algorithms solved the efficiency problem by adopting low-rank factorization models, while also proving very efficient compared to other methods. As such, the study represents a significant step forward in the challenge of developing an efficient link-prediction model.

Another important problem that needs to be addressed concerns the accuracy of link prediction approaches. developed an innovative link-prediction method that aimed to improve the accuracy of link predictions. The experimental findings revealed that the proposed approach outperformed many other methods in terms of accuracy and scalability and the associated runtime was significantly less than that observed in previous studies. However, although the proposed approach did enhance link prediction accuracy, it remains untested on large network data, which may see its efficiency become undermined.

6.About the dataset:

We'll work on a graph dataset in which nodes are Facebook pages of popular food joints and well renowned chefs from across the world. If any two pages(nodes) like each other, then there is an edge(link) between them. **Link:** http://networkrepository.com/fb-pages-food.php

Network Data Statistics	
Nodes	620
Edges	2.1K
Density	0.0108969
Maximum degree	132
Minimum degree	1
Average degree	6
Assortativity	-0.0322034
Number of triangles	8.8K
Average number of triangles	14
Maximum number of triangles	461
Average clustering coefficient	0.330897
Fraction of closed triangles	0.222641
Maximum k-core	12
Lower bound of Maximum Clique	10

Network Data Statistics

7.Implementation:

We will first import all the necessary libraries and modules:

In [1]:
```
import pandas as pd
import numpy as np
import random
import networkx as nx
from tqdm import tqdm
import re
import matplotlib.pyplot as plt
import matplotlib.colors as mcolors
```

Let's load the Facebook pages as the nodes and mutual likes between the pages as the edges:

In [2]:
```
# load nodes details
with open("fb-pages-food.nodes",encoding="cp437", errors='ignore') as f:
    fb_nodes = f.read().splitlines()

# load edges (or links)
with open("fb-pages-food.edges",encoding="cp437", errors='ignore') as f:
    fb_links = f.read().splitlines()

len(fb_nodes), len(fb_links)
```

Out[2]: (621, 2102)

7.1 Analysis:

We have 620 nodes and 2,102 links. Let's now create a dataframe of all the nodes. Every row of this dataframe represents a link formed by the nodes in the columns 'node_1' and 'node_2', respectively:

In [3]:
```
# captture nodes in 2 separate lists
node_list_1 = []
node_list_2 = []

for i in tqdm(fb_links):
  node_list_1.append(i.split(',')[0])
  node_list_2.append(i.split(',')[1])

fb_df = pd.DataFrame({'node_1': node_list_1, 'node_2': node_list_2})
fb_df.head()
```

```
100%|██████████| 2102/2102 [00:00<00:00, 381812.26it/s]
```

Out[3]:

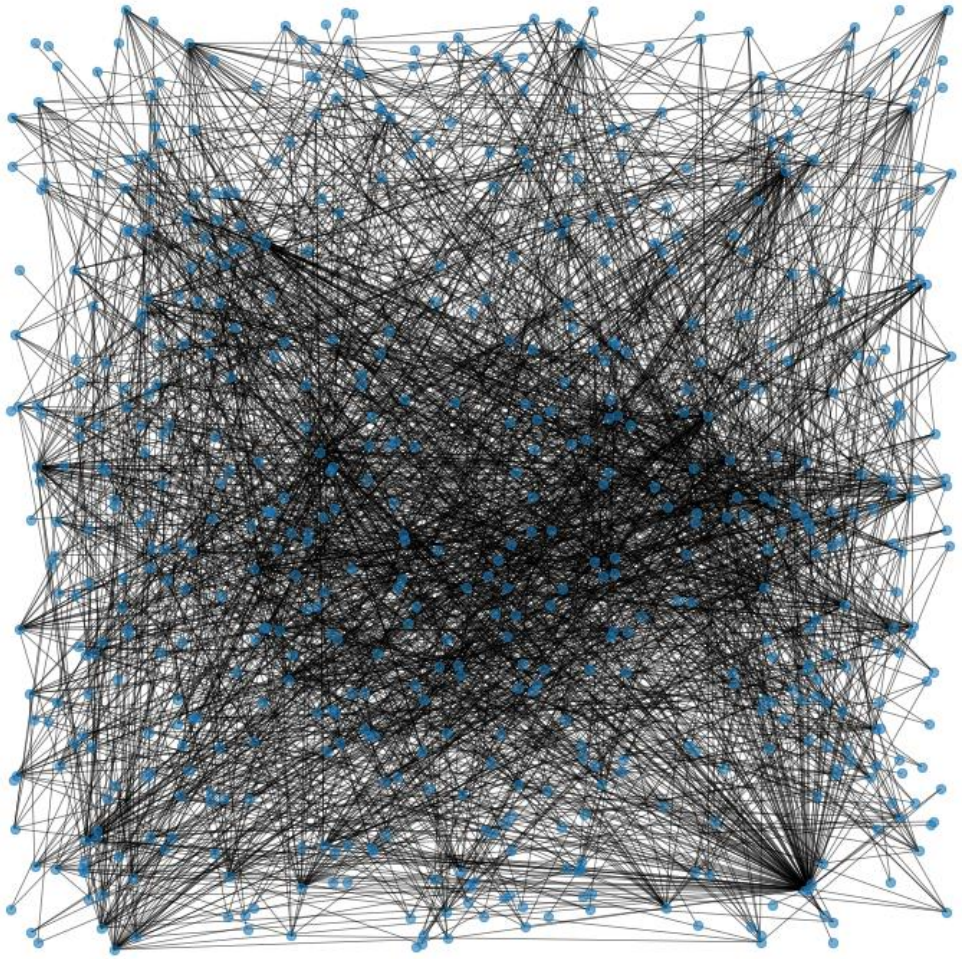
	node_1	node_2
0	0	276
1	0	58
2	0	132
3	0	603
4	0	398

The nodes ‘276’, ’58’, ‘132’, ‘603’, and ‘398’ form links with the node ‘0’. We can easily represent this arrangement of Facebook pages in the form of a graph:

In [4]:
```
# create graph
G = nx.from_pandas_edgelist(fb_df, "node_1", "node_2", create_using=nx.Graph())
print(nx.info(G))
```

```
Name:
Type: Graph
Number of nodes: 620
Number of edges: 2102
Average degree:   6.7806
```

In [5]:
```
# plot graph
plt.figure(figsize=(10,10))

pos = nx.random_layout(G, seed=23)
nx.draw(G, with_labels=False,  pos = pos, node_size = 40, alpha = 0.6, width = 0.7)

plt.show()
```



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.

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.

```
In [6]: nx.degree(G, '49')
Out[6]: 6

In [7]: nx.average_clustering(G)
Out[7]: 0.3308970263553271

In [8]: nx.shortest_path(G, '39', '49')
Out[8]: ['39', '576', '155', '293', '49']

In [9]: T = nx.bfs_tree(G, '49')
        plt.figure(figsize=(20,20))
        nx.spring_layout(T,seed=23)
        nx.draw_networkx(T)
        plt.show()
```

Distance

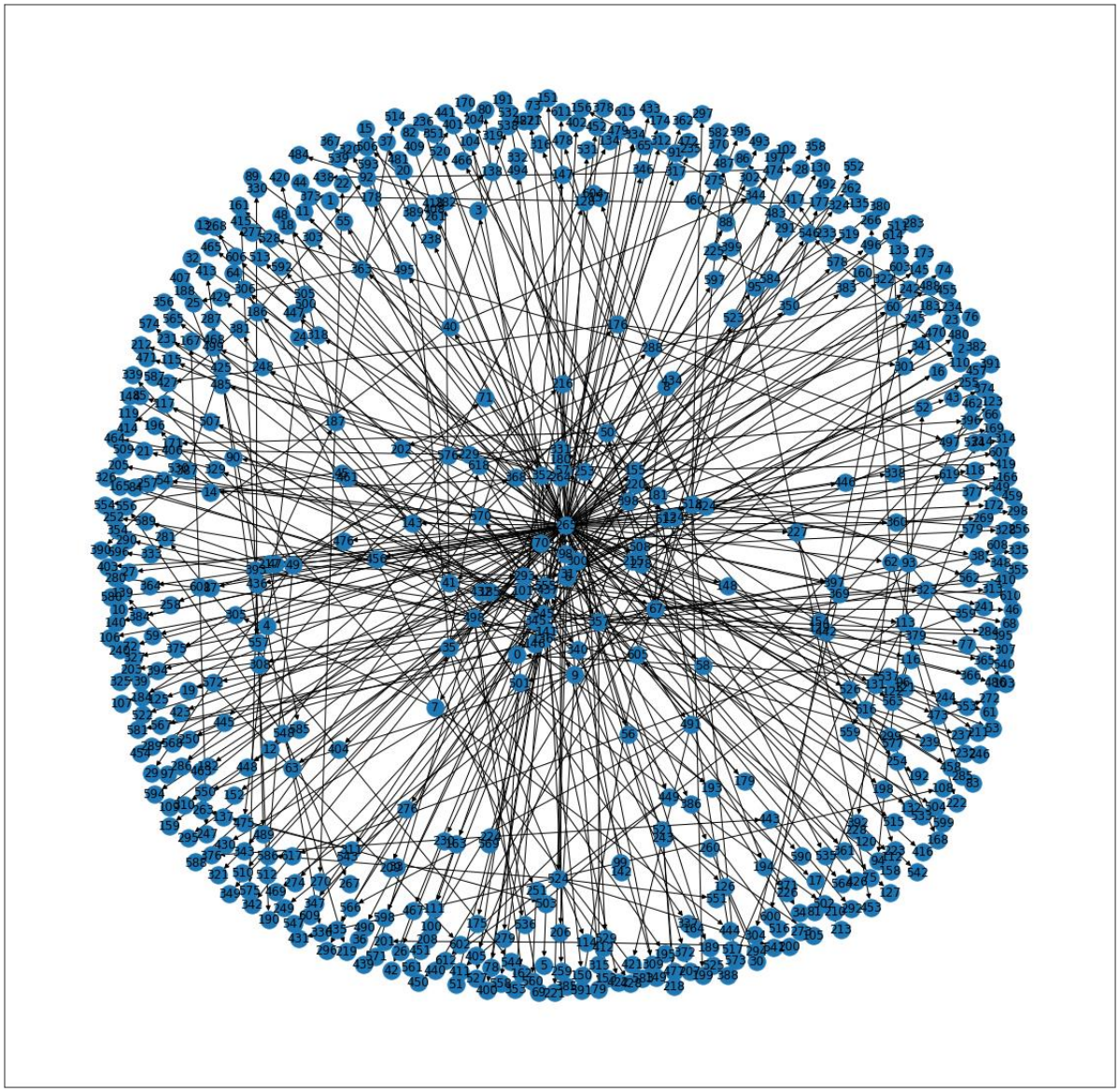
We can also determine the shortest path between two nodes and its length in NetworkX using `nx.shortest_path(Graph, Node1, Node2)` `nx.shortest_path_length(Graph, Node1, Node2)` functions respectively.

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

Eccentricity

Eccentricity of a node A is defined as the largest distance between A and all other nodes:

```
In [10]: nx.eccentricity(G,'49')
Out[10]: 12
```



Network Influencers

In this section, we will learn how to find the most important nodes (individuals) in the network. These parameters are called as **centrality measures**.

Remember that popular student from your high school or the schools top baseball player. These were the people who had the power to make your high school experience hell or heaven. What gave them this power? **Centrality Measures** can help us in identifying popularity, most liked, and biggest influencers within the network.

Degree Centrality

The people most popular or more liked usually are the ones who have more friends. Degree centrality is a measure of the number of connections a particular node has in the network. It is based on the fact that important nodes have many connections:

In [11]:
```
d=nx.degree_centrality(G)
fb_df = pd.DataFrame({'node': node, 'degree centrality': d[node]} for node in d)
fb_df.head(n=25)
```

Out[11]:

	node	degree centrality
0	0	0.009693
1	276	0.017771
2	58	0.024233
3	132	0.003231
4	603	0.043619
5	398	0.014540
6	555	0.027464
7	1	0.003231
8	265	0.216478
9	611	0.075929
10	2	0.004847
11	182	0.050081
12	345	0.017771
13	3	0.016165

Closeness Centrality

Core idea: A central node is one that is close, on average, to other nodes:

In [12]:
```
c=nx.closeness_centrality(G)
fb_df = pd.DataFrame({'node': node, 'closeness centrality': c[node]} for node in c)
fb_df.head(n=25)
```

Out[12]:

	node	closeness centrality
0	0	0.233673
1	276	0.193196
2	58	0.273531
3	132	0.227323
4	603	0.269599
5	398	0.194349
6	555	0.206678
7	1	0.249095
8	265	0.331370
9	611	0.302838
10	2	0.251524
11	182	0.267734
12	345	0.254523
13	3	0.228582
14	608	0.230454
15	377	0.209617

Betweenness Centrality

The Betweenness Centrality is the centrality of control. It represents the frequency at which a point occurs on the geodesic (shortest paths) that connected pair of points. It quantifies how many times a particular node comes in the shortest chosen path between two other nodes. The nodes with high betweenness centrality play a significant role in the communication/information flow within the network. The nodes with high betweenness centrality can have a strategic control and influence on others. An individual at such a strategic position can influence the whole group, by either withholding or coloring the information in transmission.

In [13]:
```
b=nx.betweenness_centrality(G)
fb_df = pd.DataFrame({'node': node, 'betweenness centrality': b[node]} for node in b)
fb_df.head(n=25)
```

Out[13]:

	node	betweenness centrality
0	0	0.028408
1	276	0.006052
2	58	0.034414
3	132	0.000265
4	603	0.016790
5	398	0.011679
6	555	0.055709
7	1	0.000000
8	265	0.349908
9	611	0.034436
10	2	0.000000
11	182	0.008884

Eigenvector Centrality

It is not just how many individuals one is connected too, but the type of people one is connected with that can decide the importance of a node. In Delhi Roads whenever the traffic police capture a person for breaking the traffic rule, the first sentence that traffic police hears is "Do you know whom I am related to?".

In [14]:
```
e= nx.eigenvector_centrality(G)
fb_df = pd.DataFrame({'node': node, 'eigen_vector centrality': e[node]} for node in e)
fb_df.head(n=25)
```

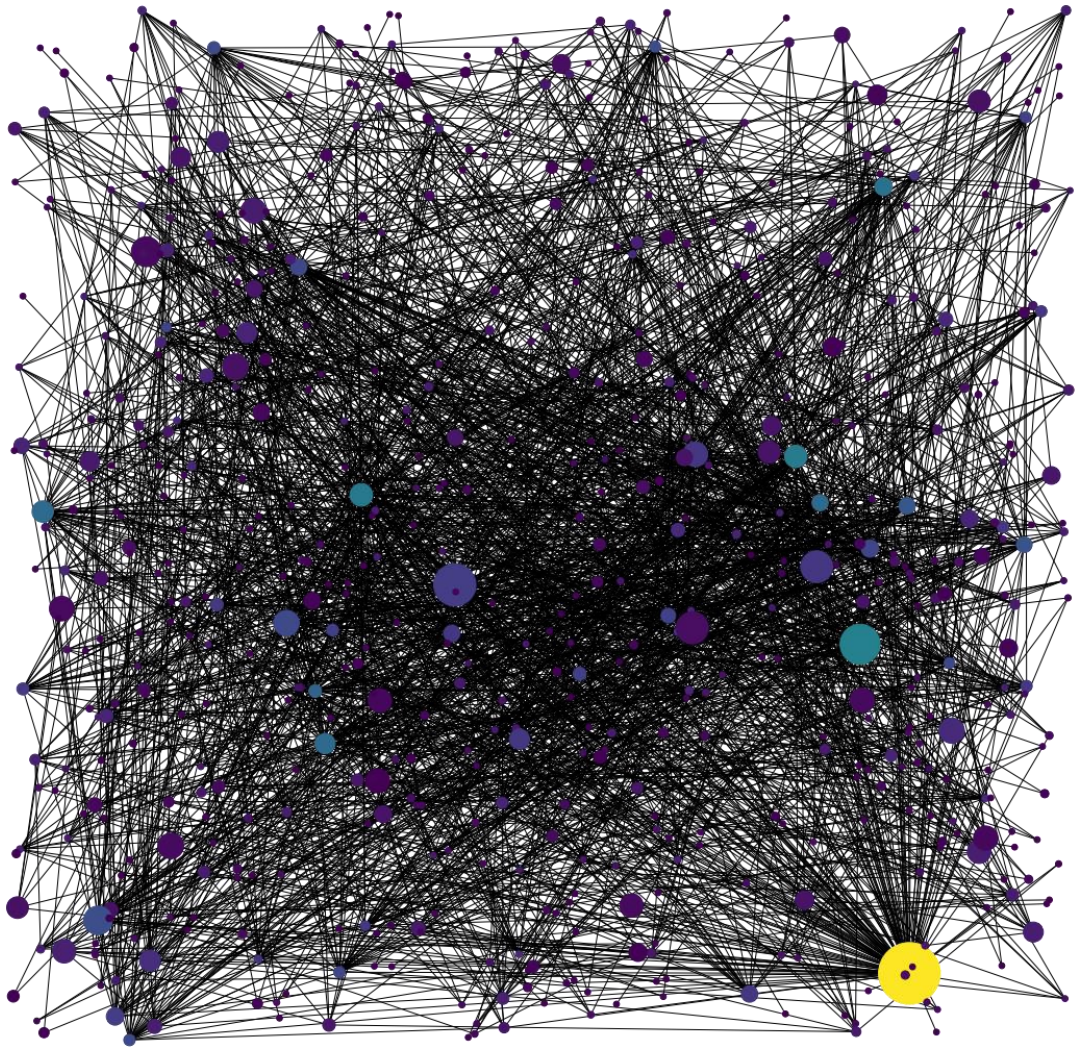
Out[14]:

	node	eigen_vector centrality
0	0	0.008551
1	276	0.000539
2	58	0.075996
3	132	0.006313
4	603	0.116762
5	398	0.000514
6	555	0.002373
7	1	0.021565
8	265	0.325752
9	611	0.184939
10	2	0.020565

7.2Visualization:

We can also visualize the network such that the node color varies with **Degree** and node size with **Betweenness Centrality**. The code to do this is:

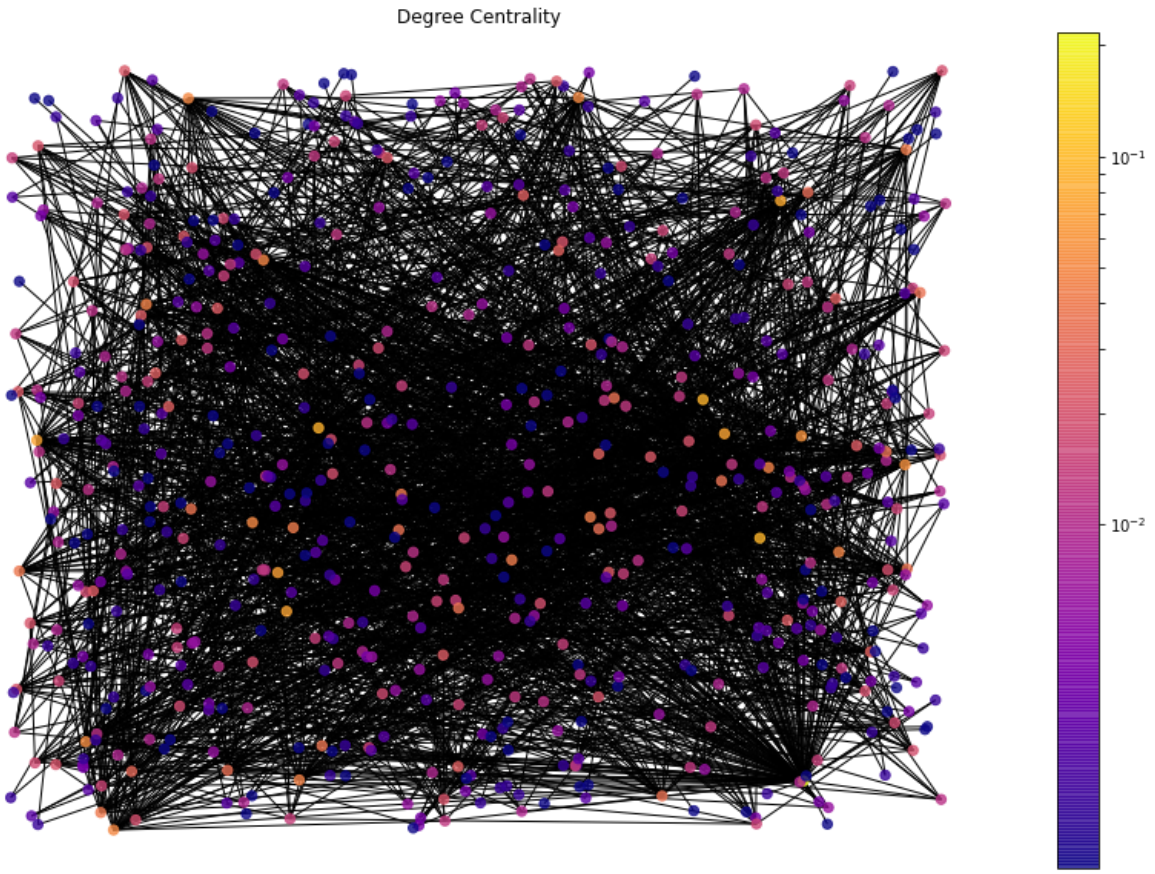
```
In [15]: pos = nx.random_layout(G, seed=23)
         betCent = nx.betweenness_centrality(G, normalized=True, endpoints=True)
         node_color = [20000.0 * G.degree(v) for v in G]
         node_size =  [v * 10000 for v in betCent.values()]
         plt.figure(figsize=(20,20))
         nx.draw_networkx(G, pos=pos, with_labels=False,
                          node_color=node_color,
                          node_size=node_size )
         plt.axis('off')
```



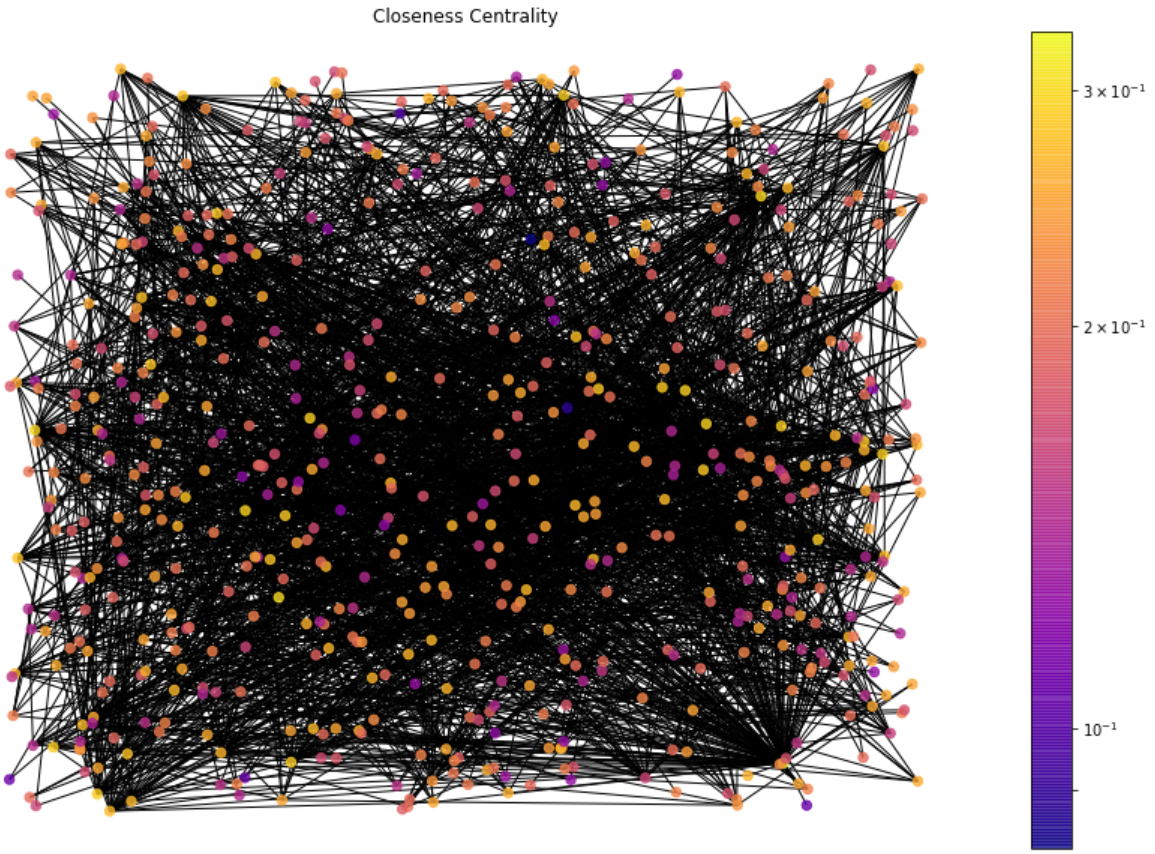
```
In [16]: sorted(betCent, key=betCent.get, reverse=True)[:5]
Out[16]: ['265', '31', '518', '618', '35']

In [17]: sorted(e, key=e.get, reverse=True)[:5]
Out[17]: ['265', '90', '340', '67', '56']

In [18]: sorted(c, key=c.get, reverse=True)[:5]
Out[18]: ['265', '611', '70', '90', '56']
```

We can see that some nodes are common between Degree Centrality, which is a measure of degree, and Betweenness Centrality which controls the information flow. It is natural that nodes that are more connected also lie on shortest paths between other nodes. The node **265** is an important node as it is crucial according to all three centrality measures that we had considered:

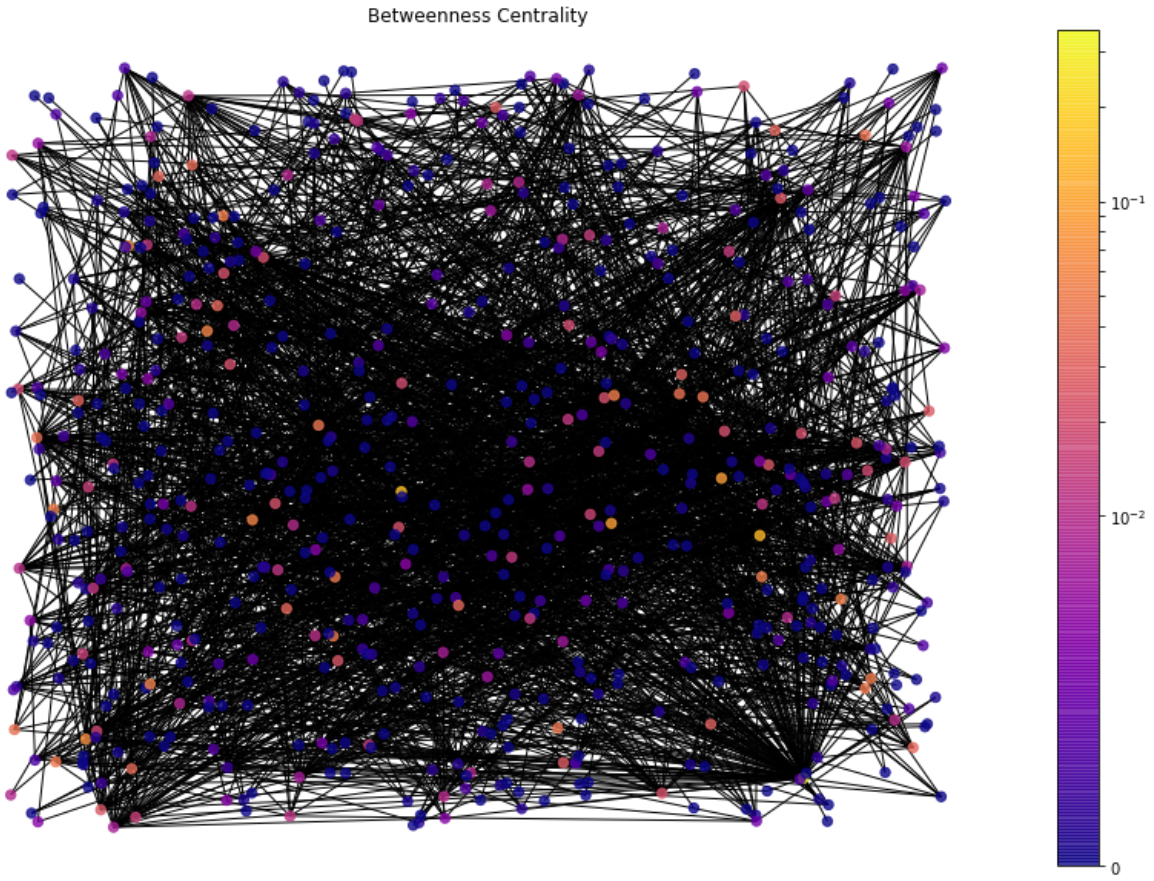
```
In [19]: %matplotlib inline
         def draw(G, pos, measures, measure_name):
             plt.figure(figsize=(15,10))
             nodes = nx.draw_networkx_nodes(G, pos, node_size=40,alpha=0.8, cmap=plt.cm.plasma,
                                            node_color=list(measures.values()),
                                            nodelist=measures.keys())
             nodes.set_norm(mcolors.SymLogNorm(linthresh=0.01, linscale=1, base=10))
             # labels = nx.draw_networkx_labels(G, pos)
             edges = nx.draw_networkx_edges(G, pos)

             plt.title(measure_name)
             plt.colorbar(nodes)
             plt.axis('off')

             plt.show()

In [20]: fb_df = pd.DataFrame({'node_1': node_list_1, 'node_2': node_list_2})
         fb_df.head()
         G = nx.from_pandas_edgelist(fb_df, "node_1", "node_2", create_using=nx.Graph())
         pos = nx.random_layout(G, seed=23)

In [21]: draw(G, pos, nx.degree_centrality(G), 'Degree Centrality')
```



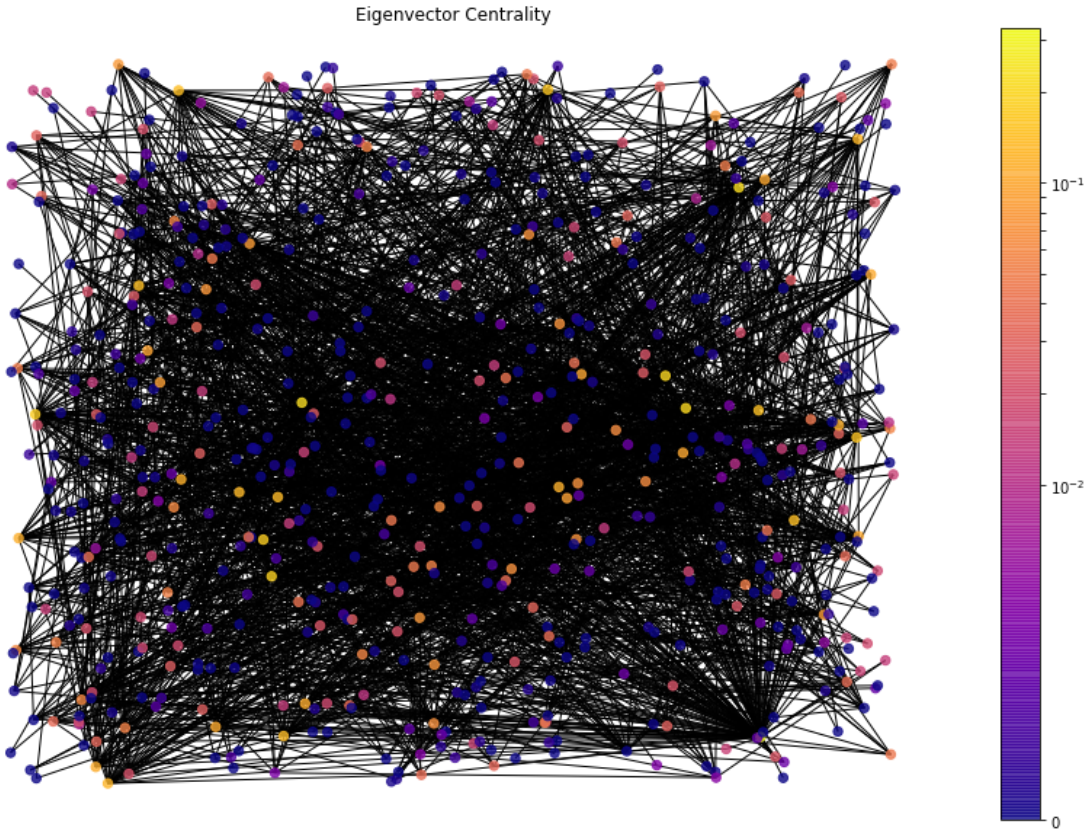
In [22]:
```
draw(G, pos, nx.closeness_centrality(G), 'Closeness Centrality')
```



In [23]:
```
draw(G, pos, nx.betweenness_centrality(G), 'Betweenness Centrality')
```



In [24]:
```
draw(G, pos, nx.eigenvector_centrality(G), 'Eigenvector Centrality')
```



```
In [25]: dict={'Closeness Centrality':sorted(c, key=c.get, reverse=True)[:25],
               'Betweenness Centrality':sorted(betCent, key=betCent.get, reverse=True)[:25],
               'Eigen Vector Centrality':sorted(e, key=e.get, reverse=True)[:25]}
         df = pd.DataFrame(dict)
         df.head(n=25)
```

Out[25]:

	Closeness Centrality	Betweenness Centrality	Eigen Vector Centrality
0	265	265	265
1	611	31	90
2	70	518	340
3	90	618	67
4	56	35	56
5	217	216	611
6	505	498	89
7	67	101	70
8	340	217	317
9	248	148	505
10	41	555	288
11	317	449	9
12	87	434	229

7.3 Dataset Preparation for Model Building

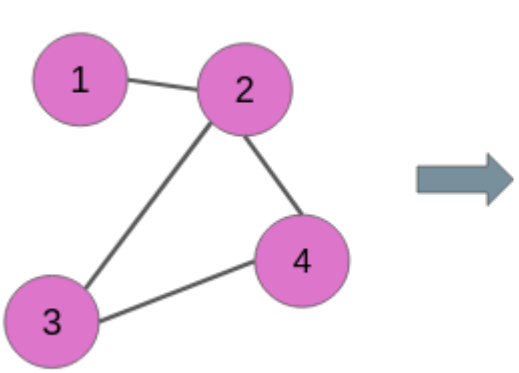
We need to prepare the dataset from an undirected graph. This dataset will have features of node pairs and the target variable would be binary in nature, indicating the presence of links (or not).

Retrieve Unconnected Node Pairs – Negative Samples

We have already understood that to solve a link prediction problem, we have to prepare a dataset from the given graph. **A major part of this dataset is the negative samples or the unconnected node pairs**. In this section, I will show you how we can extract the unconnected node pairs from a graph.

First, we will create an **adjacency matrix** to find which pairs of nodes are not connected.

For example, the adjacency of the graph below is a square matrix in which the rows and columns are represented by the nodes of the graph:



	1	2	3	4
1	0	1	0	0
2	1	0	1	1
3	0	1	0	1
4	0	1	1	0

Adjacency Matrix

The links are denoted by the values in the matrix. 1 means there is a link between the node pair and 0 means there is a link between the node pair. For instance, nodes 1 and 3 have 0 at their cross-junction in the matrix and these nodes also have no edge in the graph above.We will use this property of the adjacency matrix to find all the unconnected node pairs from the original graph **G**:

```
In [26]: # combine all nodes in a list
         node_list = node_list_1 + node_list_2

         # remove duplicate items from the list
         node_list = list(dict.fromkeys(node_list))

         # build adjacency matrix
         adj_G = nx.to_numpy_matrix(G, nodelist = node_list)

In [27]: adj_G.shape

Out[27]: (620, 620)
```

As you can see, it is a square matrix. Now, we will traverse the adjacency matrix to find the positions of the zeros. Please note that we don't have to go through the entire matrix. The values in the matrix are the same above and below the diagonal, as you can see below:

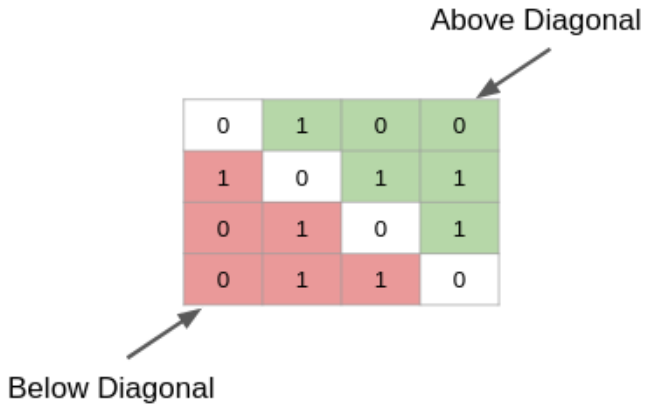


Fig. 3

We can either search through the values above the diagonal (green part) or the values below (red part). Let's search the diagonal values for zero:

```
In [29]: # get unconnected node-pairs
         all_unconnected_pairs = []

         # traverse adjacency matrix
         offset = 0
         for i in tqdm(range(adj_G.shape[0])):
           for j in range(offset,adj_G.shape[1]):
             if i != j:
               if nx.shortest_path_length(G, str(i), str(j)) <=2:
                 if adj_G[i,j] == 0:
                   all_unconnected_pairs.append([node_list[i],node_list[j]])

           offset = offset + 1

         100%|██████████| 620/620 [00:17<00:00, 34.76it/s]

In [30]: len(all_unconnected_pairs)

Out[30]: 19018
```

We have 19,018 unconnected pairs. These node pairs will act as negative samples during the training of the link prediction model. Let's keep these pairs in a dataframe:

```
In [31]: node_1_unlinked = [i[0] for i in all_unconnected_pairs]
         node_2_unlinked = [i[1] for i in all_unconnected_pairs]

         data = pd.DataFrame({'node_1':node_1_unlinked,
                              'node_2':node_2_unlinked})

         # add target variable 'link'
         data['link'] = 0
```

7.4 Remove Links from Connected Node Pairs – Positive Samples

As we discussed above, we will randomly drop some of the edges from the graph. However, randomly removing edges may result in cutting off loosely connected nodes and fragments of the graph. This is something that we have to take care of. **We have to make sure that in the process of dropping edges, all the nodes of the graph should remain connected.**

In the code block below, we will first check if dropping a node pair results in the splitting of the graph (number_connected_components > 1) or reduction in the number of nodes. If both things do not happen, then we drop that node pair and repeat the same process with the next node pair.

Eventually, we will get a list of node pairs that can be dropped from the graph and all the nodes would still remain intact:

```
In [32]: initial_node_count = len(G.nodes)

         fb_df_temp = fb_df.copy()

         # empty list to store removable links
         omissible_links_index = []

         for i in tqdm(fb_df.index.values):

           # remove a node pair and build a new graph
           G_temp = nx.from_pandas_edgelist(fb_df_temp.drop(index = i), "node_1", "node_2", cre

           # check there is no spliting of graph and number of nodes is same
           if (nx.number_connected_components(G_temp) == 1) and (len(G_temp.nodes) == initial_r
             omissible_links_index.append(i)
             fb_df_temp = fb_df_temp.drop(index = i)

         100%|██████████| 2102/2102 [00:15<00:00, 133.07it/s]

In [33]: len(omissible_links_index)

Out[33]: 1483
```

We have over 1400 links that we can drop from the graph. These dropped edges will act as positive training examples during the link prediction model training.

Data for Model Training

Next, we will append these removable edges to the dataframe of unconnected node pairs. Since these new edges are positive samples, they will have a target value of '1':

```
In [34]: # create dataframe of removable edges
         fb_df_ghost = fb_df.loc[omissible_links_index]

         # add the target variable 'link'
         fb_df_ghost['link'] = 1

         data = data.append(fb_df_ghost[['node_1', 'node_2', 'link']], ignore_index=True)

In [35]: data['link'].value_counts()

Out[35]: 0    19018
         1     1483
         Name: link, dtype: int64
```

It turns out that this is highly imbalanced data. The ratio of link vs no link is just close to 8%. In the next section, we will extract features for all these node pairs.

7.5 Feature Extraction

We will use the *node2vec* algorithm to extract node features from the graph after dropping the links. So, let's first create a new graph after dropping the removable links:

```
In [36]: # drop removable edges
         fb_df_partial = fb_df.drop(index=fb_df_ghost.index.values)

         # build graph
         G_data = nx.from_pandas_edgelist(fb_df_partial, "node_1", "node_2", create_using=nx.Graph())
```

Next, we will install the *node2vec* library. It is quite similar to the DeepWalk algorithm. However, it involves biased random walks. To know more about *node2vec*, you should definitely check out this paper **node2vec: Scalable Feature Learning for Networks**.

For the time being, just keep in mind *node2vec* is used for vector representation of nodes of a graph. Let's install it:

```
In [37]: !pip install node2vec
```

```
Collecting node2vec
  Downloading node2vec-0.3.2-py3-none-any.whl (6.5 kB)
Requirement already satisfied: joblib>=0.13.2 in c:\users\alpanshu kataria\anaconda3\lib\site-packages (from node2ve
Requirement already satisfied: networkx in c:\users\alpanshu kataria\anaconda3\lib\site-packages (from node2vec) (2.
Collecting gensim
  Downloading gensim-3.8.3-cp38-cp38-win_amd64.whl (24.2 MB)
Requirement already satisfied: numpy in c:\users\alpanshu kataria\anaconda3\lib\site-packages (from node2vec) (1.18.
Requirement already satisfied: tqdm in c:\users\alpanshu kataria\anaconda3\lib\site-packages (from node2vec) (4.47.0
Requirement already satisfied: decorator>=4.3.0 in c:\users\alpanshu kataria\anaconda3\lib\site-packages (from netwo
ec) (4.4.2)
Collecting Cython==0.29.14
  Downloading Cython-0.29.14-cp38-cp38-win_amd64.whl (1.7 MB)
Requirement already satisfied: six>=1.5.0 in c:\users\alpanshu kataria\anaconda3\lib\site-packages (from gensim->nod
5.0)
```

```
In [38]: from node2vec import Node2Vec

         # Generate walks
         node2vec = Node2Vec(G_data, dimensions=100, walk_length=16, num_walks=50)

         # train node2vec model
         n2w_model = node2vec.fit(window=7, min_count=1)
```

```
Computing transition probabilities: 100%|██████████| 620/620 [00:00<00:00, 6521.67it/s]
Generating walks (CPU: 1): 100%|██████████| 50/50 [00:32<00:00,  1.52it/s]
```

Next, we will apply the trained *node2vec* model on each and every node pair in the dataframe 'data'. To compute the features of a pair or an edge, we will add up the features of the nodes in that pair:

```
In [39]: x = [(n2w_model[str(i)]+n2w_model[str(j)]) for i,j in zip(data['node_1'], data['node_2'])]
```

```
<ipython-input-39-621e060f11b3>:1: DeprecationWarning: Call to deprecated `__getitem__` (Metho
elf.wv.__getitem__() instead).
  x = [(n2w_model[str(i)]+n2w_model[str(j)]) for i,j in zip(data['node_1'], data['node_2'])]
```

7.6 Building our Link Prediction Model

To validate the performance of our model, we should split our data into two parts – one for training the model and the other to test the model's performance:

```
In [42]: from sklearn.linear_model import LogisticRegression
         from sklearn.metrics import classification_report, roc_auc_score
         from sklearn.model_selection import train_test_split
         from sklearn.metrics import confusion_matrix
```

```
In [43]: xtrain, xtest, ytrain, ytest = train_test_split(np.array(x), data['link'],
                                                       test_size = 0.3,
                                                       random_state = 35)
```

7.7 Logistic regression model:

Let's fit a logistic regression model first:

In [44]:
```
lr = LogisticRegression(class_weight="balanced")

lr.fit(xtrain, ytrain)
```

```
C:\Users\Alpanshu Kataria\anaconda3\lib\site-packages\sklearn\linear_model\_logisti
o converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Out[44]:
```
LogisticRegression(class_weight='balanced')
```

In [45]:
```
predictions = lr.predict_proba(xtest)
```

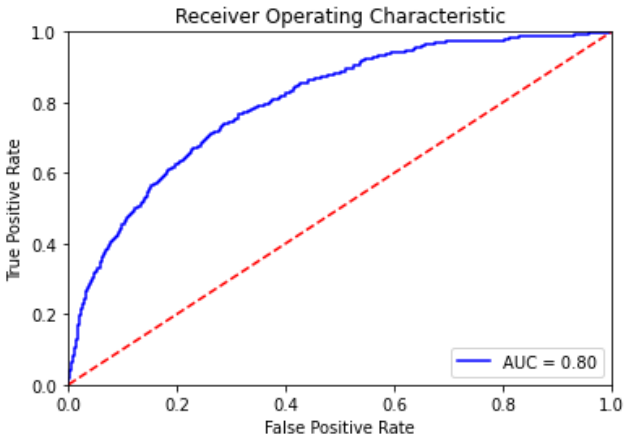
We will use the AUC-ROC score to check our model's performance.

In [43]:
```
roc_auc_score(ytest, predictions[:,1])
```

Out[43]:
```
0.7996240149564944
```

In [48]:
```
import sklearn.metrics as metrics
# calculate the fpr and tpr for all thresholds of the classification
probs = lr.predict_proba(xtest)
preds = probs[:,1]
fpr, tpr, threshold = metrics.roc_curve(ytest, preds)
roc_auc = metrics.auc(fpr, tpr)
```

In [49]:
```
import matplotlib.pyplot as plt
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



AUC ROC Curve

We get a score of 0.80 using a logistic regression model. Let's see if we can get a better score by using a more complex model.

7.8 LightGBM vs Logistic regression

The development of Logistic regression started way back. The logistic regression as a general statistical model was originally developed and popularized primarily by Joseph Berkson,[3] beginning in Berkson (1944 .Logistic regression has become a de-facto algorithm ,simply because it is extremely powerful. But given lots and lots of data, it takes a long time to train.

Here comes…. Light GBM into the picture.

What is Light GBM?

Light GBM is a fast, distributed, high-performance gradient boosting framework based on decision tree algorithm, used for ranking, classification and many other machine learning tasks.

Since it is based on decision tree algorithms, it splits the tree leaf wise with the best fit whereas other boosting algorithms split the tree depth wise or level wise rather than leaf-wise. So when growing on the same leaf in Light GBM, the leaf-wise algorithm can reduce more loss than the level-wise algorithm and hence results in much better accuracy which can rarely be achieved by any of the existing algorithms.

Below is a diagrammatic representation by the makers of the Light GBM to explain the difference clearly.

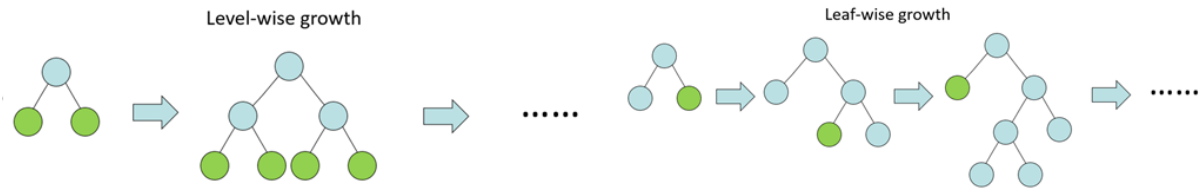


Fig. 4

Advantages of Light GBM

1. **Faster training speed and higher efficiency**: Light GBM use histogram based algorithm i.e it buckets continuous feature values into discrete bins which fasten the training procedure.

2. **Lower memory usage:** Replaces continuous values to discrete bins which result in lower memory usage.

3. **Better accuracy than any other algorithm:** It produces much more complex trees by following leaf wise split approach rather than a level-wise approach which is the main factor in achieving higher accuracy. However, it can sometimes lead to overfitting which can be avoided by setting the max_depth parameter.

4. **Compatibility with Large Datasets:** It is capable of performing equally good with large datasets with a significant reduction in training time as compared to Logistic regression.

Let's try out LightGBM:

```
In [49]: !pip3 install lightgbm
```

```
In [45]: import lightgbm as lgbm
```

```
In [46]: train_data = lgbm.Dataset(xtrain, ytrain)
         test_data = lgbm.Dataset(xtest, ytest)

         # define parameters
         parameters = {
             'objective': 'binary',
             'metric': 'auc',
             'is_unbalance': 'true',
             'feature_fraction': 0.5,
             'bagging_fraction': 0.5,
             'bagging_freq': 20,
             'num_threads' : 2,
             'seed' : 76
         }

         # train lightGBM model
         model = lgbm.train(parameters,
                            train_data,
                            valid_sets=test_data,
                            num_boost_round=1000,
                            early_stopping_rounds=20)
```

```
[152]   valid_0's auc: 0.921833
[153]   valid_0's auc: 0.921907
[154]   valid_0's auc: 0.92213
[155]   valid_0's auc: 0.92216
[156]   valid_0's auc: 0.921988
[157]   valid_0's auc: 0.92192
[158]   valid_0's auc: 0.921933
Early stopping, best iteration is:
[138]   valid_0's auc: 0.923631
```

The training stopped after the 208th iteration because we applied the early stopping criteria. Most importantly, **the model got an impressive 0.9236 AUC score on the test set**

8. Conclusion:

In this project, we have performed, as far as we know, the most comprehensive study about the link prediction method.The most important results that support specific link prediction techniques and network formation models have also been described. A comprehensive experimentation has been performed using ma particular network with different properties. It has been observed that new links can be better predicted using only local or quasi-local information in most networks. Considering indirect connections only adds noise and computational complexity to the link prediction problem. Link prediction is a relatively young research area and many open challenges remain. Further studies are required in order to understand why some methods work better or worse than others depending on the network they are applied to. Studying which network structural properties lead to better performance for each technique is an open research problem. In addition, very few techniques adapt to the global structure of the network and no technique adapts to the local structure of networks. The main difficulty when dealing with complex networks in practice is their size, which limits the kinds of techniques that can be applied.Link prediction remains an open research problem, given its importance in many applications. New techniques with better accuracy and performance tradeoffs are expected to be proposed in the forthcoming future.

9.References:

[1] Mohammad Marjan, Nazar Zaki, and Elfadil A. Mohamed (2018). “Link Prediction in Dynamic Social Networks: A Literature Review.” IEEE CiSt'18 5th Edition International IEEE Congress on Information Science and Technology, Marrakech, Morocco, October 21 - 27, 2018.

[2] Wang, P., Xu, B., Wu, Y. and Zhou, X. (2014). Link prediction in social networks: the state-of-the-art. arXiv preprint arXiv:1411.5118.

[3] Al Hasan, M. and Zaki, M. J. (2011). A survey of link prediction in social networks. In Social network data analytics (pp. 243–275). Springer US.

[4] Tylenda, T., Angelova, R. and Bedathur, S. (2009, June). Towards timeaware link prediction in evolving social networks. In Proceedings of the 3rd workshop on social network mining and analysis (p. 9). ACM.

[5] Wu S., Sun J., Tang J. Patent partner recommendation in enterprise social networks. In: Proceedings of the 6th ACM International Conference on Web Search and Data Mining (WSDM’13), Rome, Italy, 2013. 43–52