NETWORK LINK PREDICTION FOR YOUR FUTURE CONNECTION IN FACEBOOK

Project Report

CHIRAG GUPTA (19Z207)

DARWIN DEBBARMA (19Z208)

VARUN BHARDWAJ (19Z256)

SRINATH S (20z433)

VIGNESH A (20Z436)

ROHIT SONAR (19Z342)

Assignment Submission in partial fulfilment of the degree

BACHELOR OF ENGINEERING

Branch: COMPUTER SCIENCE AND ENGINEERING

Of Anna University

November 2022

PSG College of Technology

Coimbatore – 641004

1.Problem Statement:

To visualize stack overflow as a social network and do,

- Prediction model to predict future links (mutual likes) between unconnected nodes (Facebook pages).

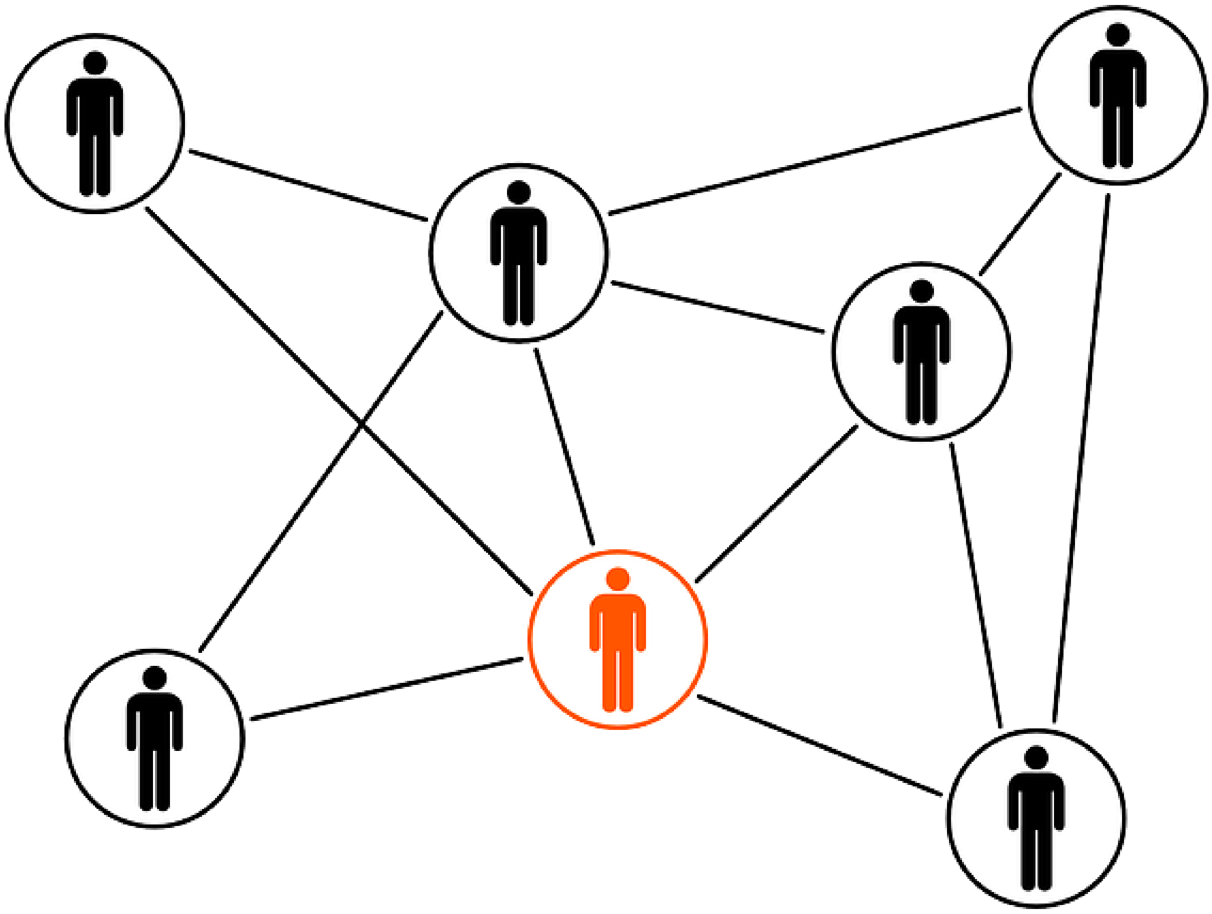
2.Overview:

A social network is essentially a representation of the relationships between social entities such as people, organizations, governments and political parties.

Interactions between these entities generate unimaginable amounts of data in the form of posts, chat messages, and tweets, like, comment, share and more. This opens a window of possibilities and use cases for us to work with, which leads us to Social Network Analytics (SNA). It can be defined as a combination of several activities that take place on social media. These activities include collecting data from online social media websites and using that data to make decisions.

What is link prediction?

Link prediction is one of the most important research topics in the field of graphs and networks. The goal of link prediction is to identify pairs of nodes that will or will not form links in the future.

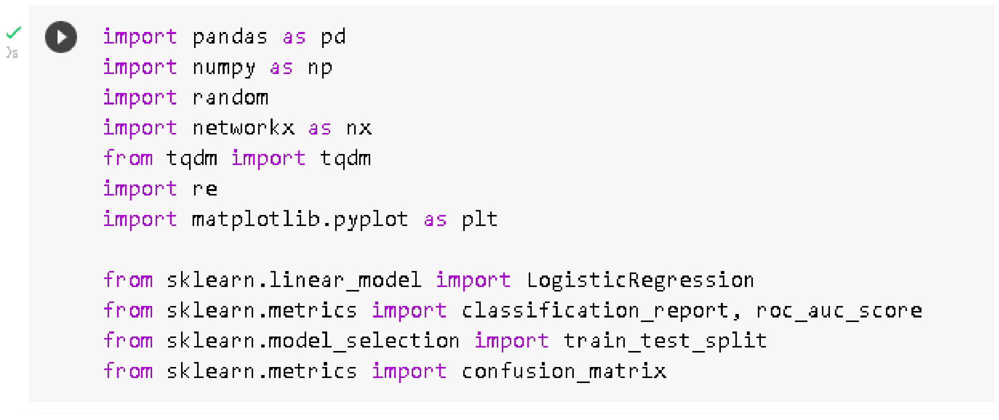


Link prediction has many uses in real-world applications. Here are some key use cases for link prediction:

1. Predict which customers are likely to purchase which products on online marketplaces such as Amazon. It helps us make better product recommendations.
2. Suggest interaction or collaboration between employees within the organization.

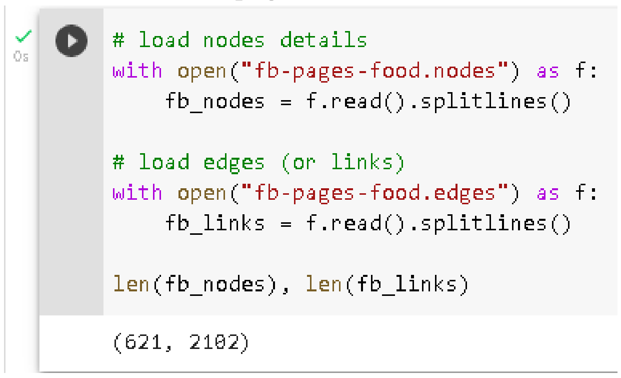
3.Implementing Link Prediction System

We will first import all the necessary libraries and modules



```
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

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
```

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

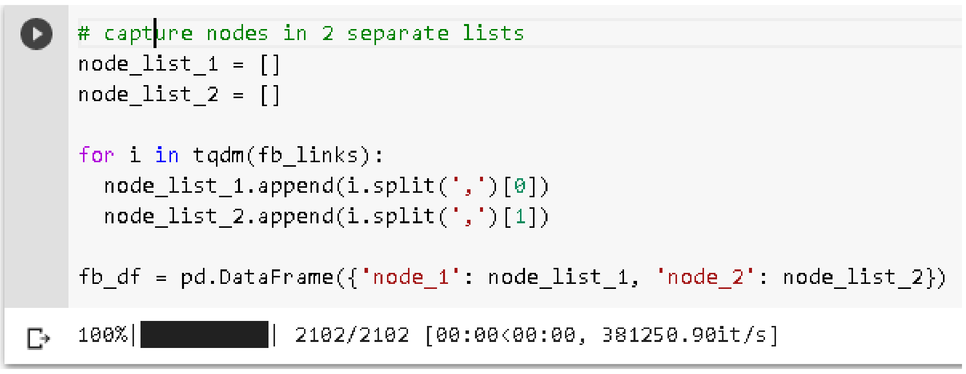


```
# load nodes details
with open("fb-pages-food.nodes") as f:
    fb_nodes = f.read().splitlines()

# load edges (or links)
with open("fb-pages-food.edges") as f:
    fb_links = f.read().splitlines()

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

```
(621, 2102)
```

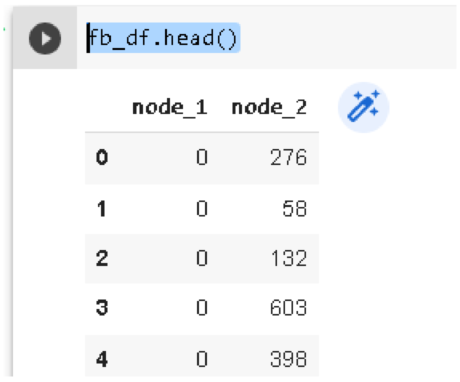
There are 620 nodes and 2,102 links.

Create a dataframe of all the nodes. Each row in this data frame represents a link formed by the nodes in the 'node_1' and 'node_2' columns.



```
# capture 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})
```

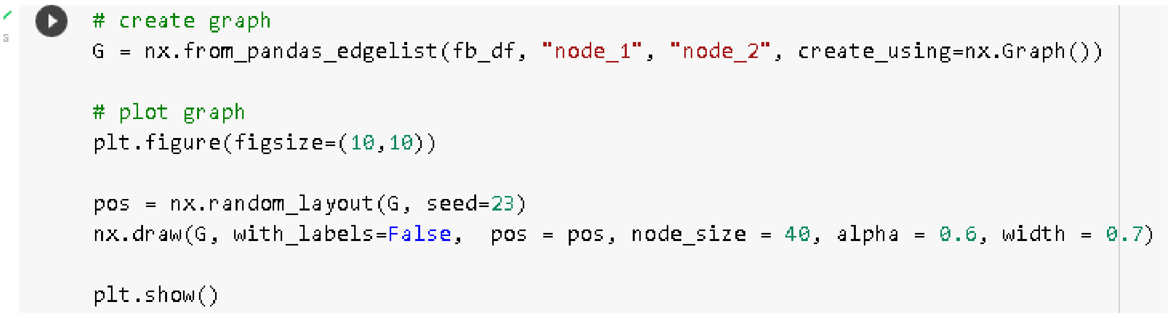
```
100%|██████████| 2102/2102 [00:00<00:00, 381250.90it/s]
```



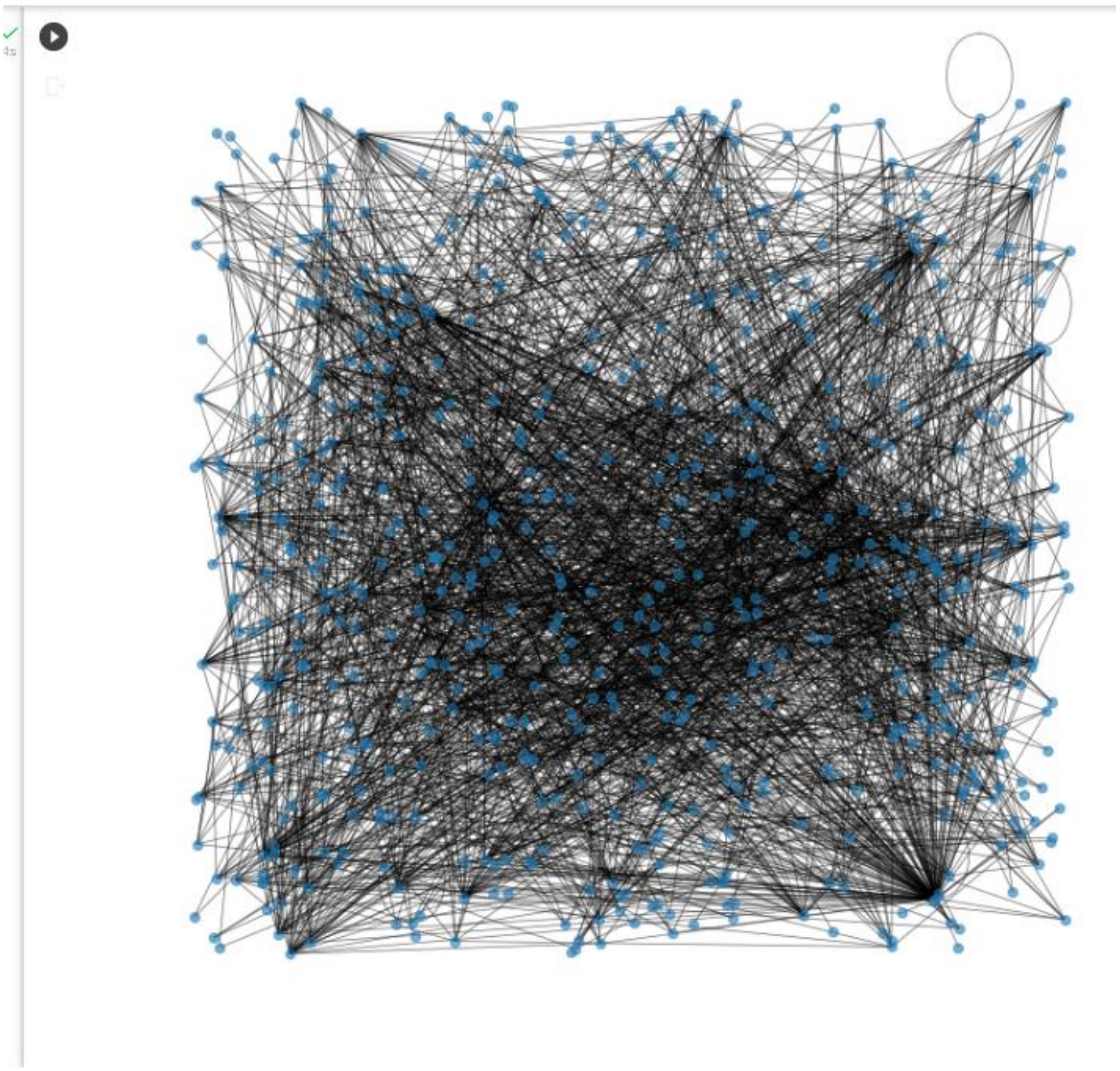
```
fb_df.head()
```

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

Nodes '276', '58', '132', '603' and '398' form links with node '0'. We can easily represent the layout of this Facebook page in the form of a diagram.



```
# create graph
G = nx.from_pandas_edgelist(fb_df, "node_1", "node_2", create_using=nx.Graph())

# 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()
```

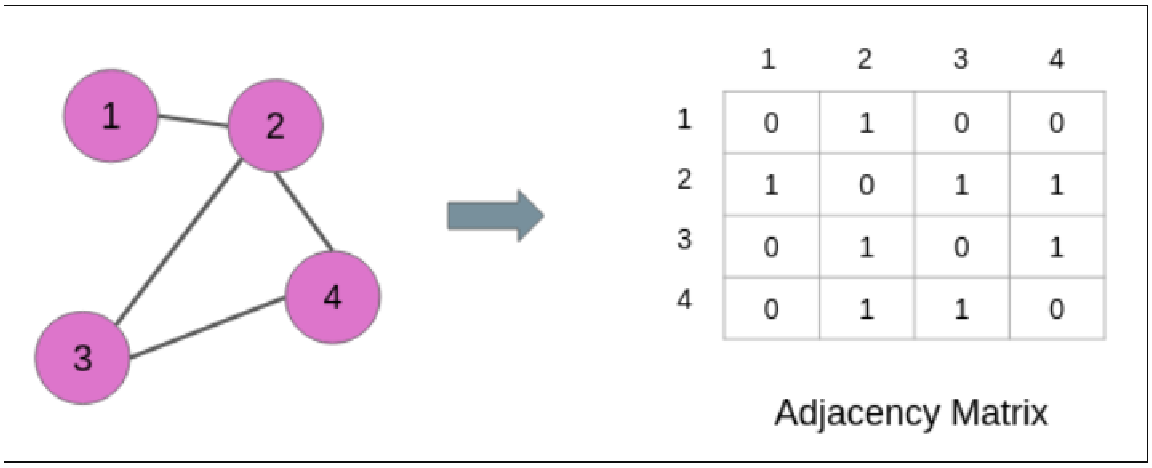


Retrieve Unconnected Node Pairs – Negative Samples

We already understood that to solve the link prediction issue, we need to create a dataset from the given chart. Many of this data set are negative samples or unconnected node pairs.

First, construct an adjacency matrix to find pairs of unconnected nodes.

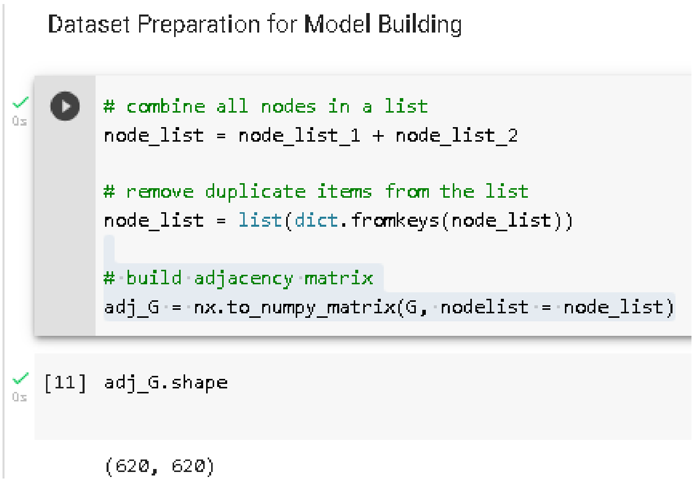
For example, the neighbourhood in the following graph is a square matrix whose rows and columns are represented by the nodes of the graph.



$$\begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 1 & 0 & 0 \\ 2 & 1 & 0 & 1 & 1 \\ 3 & 0 & 1 & 0 & 1 \\ 4 & 0 & 1 & 1 & 0 \end{array}$$

Links are identified by matrix values. 1 means there is a connection between the pair of nodes, 0 means there is a connection between the pair of nodes.
For example, nodes 1 and 3 have a 0 at the intersection of the matrices, and these nodes also have no edges in the diagram above.

We will use the property of the adjacency matrix **to find all the unconnected node** pairs from the original graph

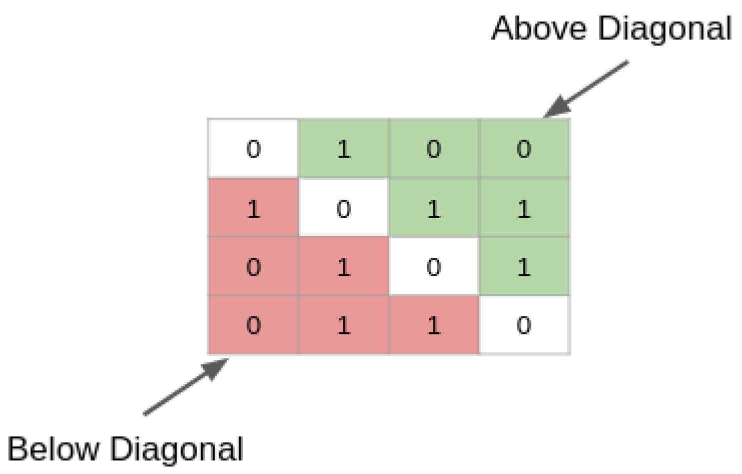


Dataset Preparation for Model Building

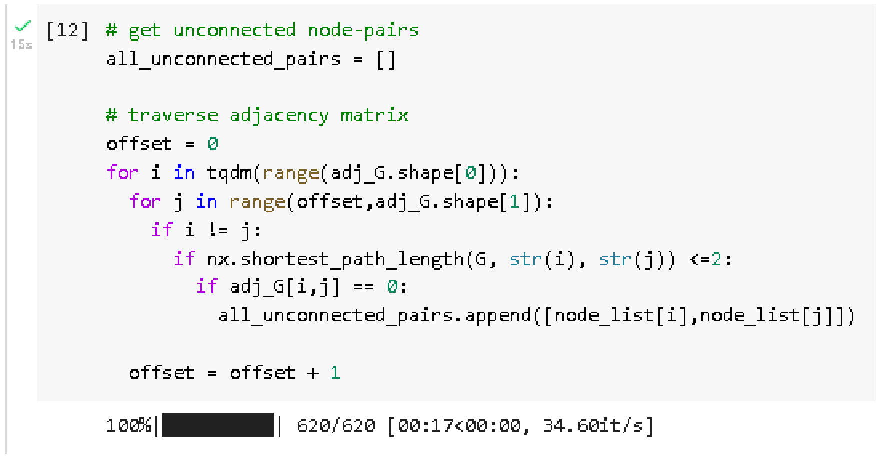
```
# combine all nodes in a list
node_list = node_list_1 + node_list_2

# 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)
```

```
[11] adj_G.shape

(620, 620)
```

As we can see, it's a square matrix. Then look at the adjacency matrix to find the position of the zero. Note that you don't have to traverse the entire matrix. The matrix values are the same above and below the diagonal, as shown below.

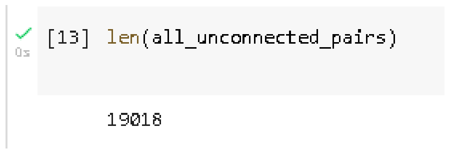


We can search for values above (green part) or below (red part) the diagonal. Let's find the **diagonal value of zero:**



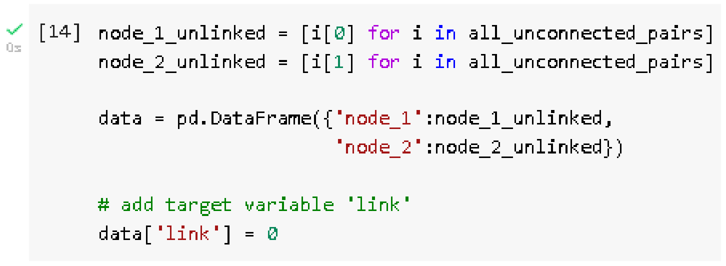
```
[12] # get unconnected node-pairs
     all_unconnected_pairs = []

     # 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, 34.60it/s]
```

Unconnected node pairs we have in our dataset:



```
[13] len(all_unconnected_pairs)

     19018
```

There are 19,018 unconnected pairs. These node pairs serve as negative samples during training of the link prediction model. Let's keep these pairs in a dataframe



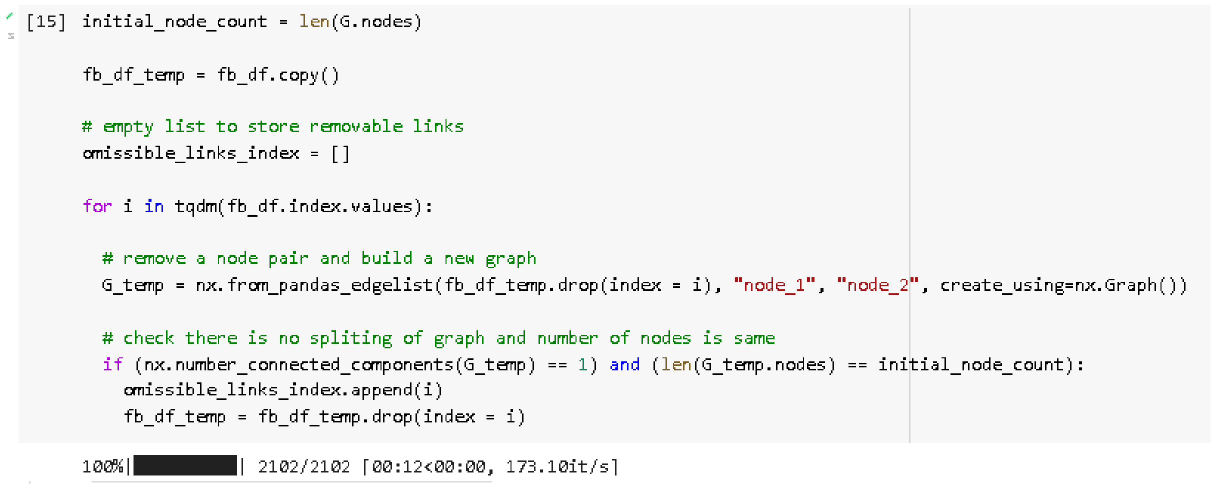
```
[14] node_1_unlinked = [i[0] for i in all_unconnected_pairs]
     node_2_unlinked = [i[1] for i in all_unconnected_pairs]

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

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

Remove Links from Connected Node Pairs – Positive Samples

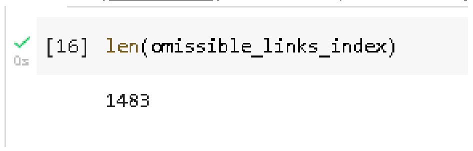
Randomly remove some edges from the graph. However, randomly removing edges can truncate loosely connected nodes and diagram fragments. It's something we need to take care of. We need to ensure that all vertices in the graph remain connected when an edge is dropped.

The following code block first checks to see if omitting a pair of nodes splits the graph (number_connected_components > 1) or reduces the number of nodes. If neither happens, drop that pair of nodes and repeat the process with the next pair of nodes.

Finally, we get the list of node pairs that can be removed from the graph. All nodes remain intact.



```
[15] 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", create_using=nx.Graph())

       # 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_node_count):
         omissible_links_index.append(i)
         fb_df_temp = fb_df_temp.drop(index = i)

     100%|██████████| 2102/2102 [00:12<00:00, 173.10it/s]
```

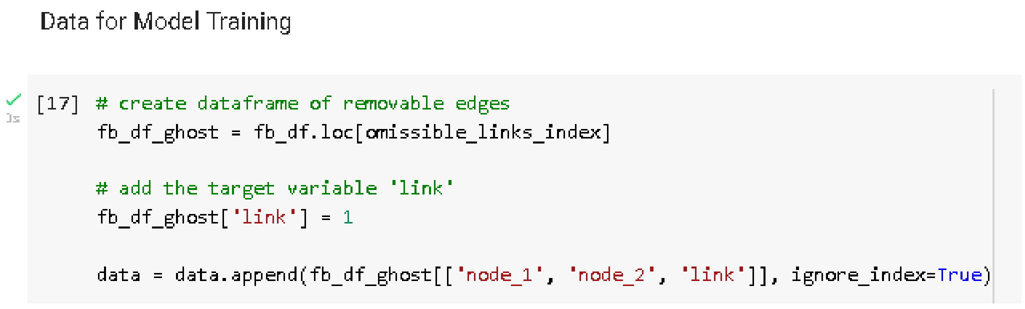


```
[16] len(omissible_links_index)

     1483
```

There are over 1400 links that can be removed from the chart. These dropped edges serve as positive training samples during training of the link prediction model.

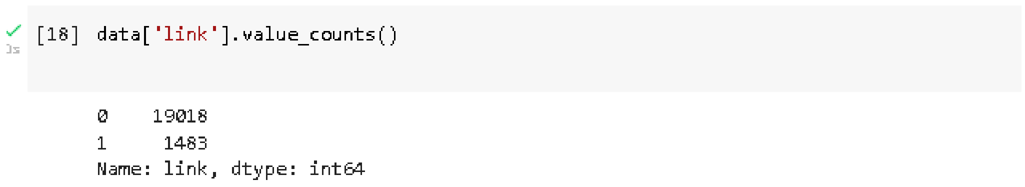
Data for Model Training

Attach these detachable edges to data frames of unconnected pairs of nodes. These new edges are positive samples, so the target value is '1'.



Data for Model Training

```
[17] # 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)
```



```
[18] data['link'].value_counts()

     0    19018
     1     1483
     Name: link, dtype: int64
```

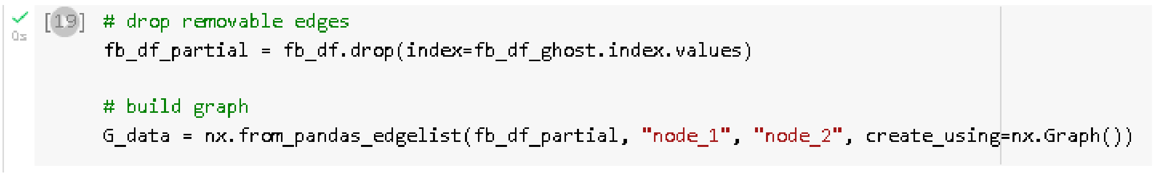
The distribution of values of the target variable:

```
0    19018
1    1483
```

We can see that this is very imbalanced data. The link/no link ratio is just under 8%.

Feature Extraction

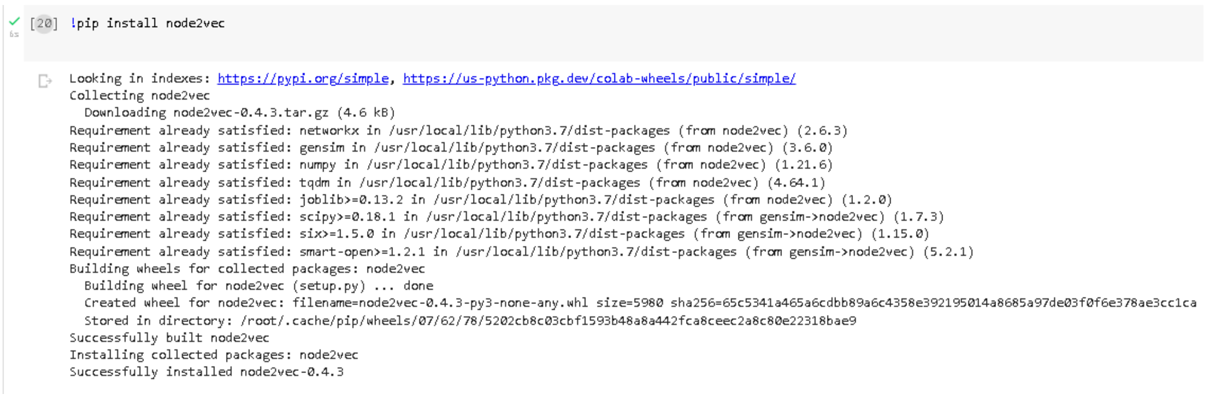
After removing the links, we use the node2vec algorithm to extract the node features from the diagram. So after removing the removable link, we first create a new diagram.



```
[19] # 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())
```

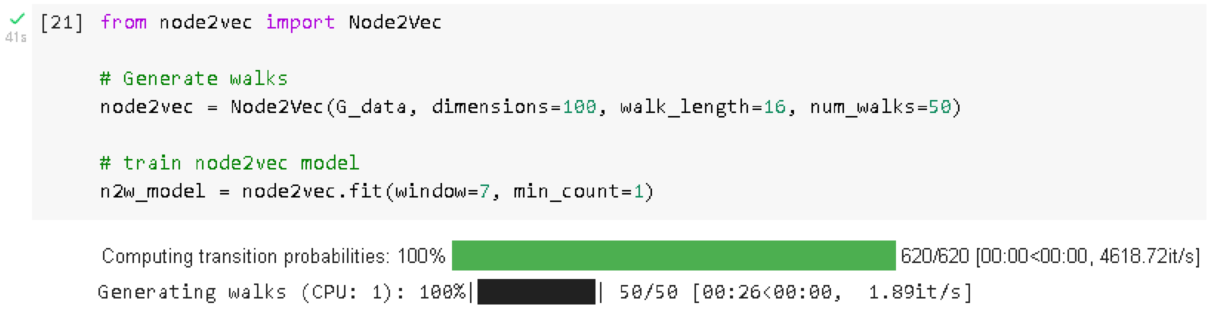
Then install the node2vec library. This is very similar to the DeepWalk algorithm. However, these are biased random walks.
node2vec is used for the vector representation of the nodes of the graph.

Install node2vec :



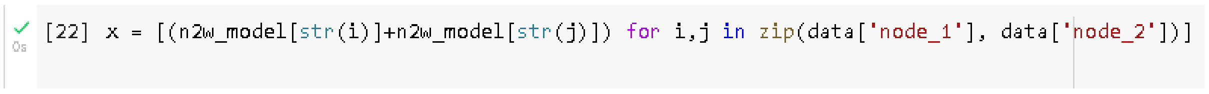
```
[20] !pip install node2vec

     Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
     Collecting node2vec
       Downloading node2vec-0.4.3.tar.gz (4.6 kB)
     Requirement already satisfied: networkx in /usr/local/lib/python3.7/dist-packages (from node2vec) (2.6.3)
     Requirement already satisfied: gensim in /usr/local/lib/python3.7/dist-packages (from node2vec) (3.6.0)
     Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from node2vec) (1.21.6)
     Requirement already satisfied: tqdm in /usr/local/lib/python3.7/dist-packages (from node2vec) (4.64.1)
     Requirement already satisfied: joblib>=0.13.2 in /usr/local/lib/python3.7/dist-packages (from node2vec) (1.2.0)
     Requirement already satisfied: scipy>=0.18.1 in /usr/local/lib/python3.7/dist-packages (from gensim->node2vec) (1.7.3)
     Requirement already satisfied: six>=1.5.0 in /usr/local/lib/python3.7/dist-packages (from gensim->node2vec) (1.15.0)
     Requirement already satisfied: smart-open>=1.2.1 in /usr/local/lib/python3.7/dist-packages (from gensim->node2vec) (5.2.1)
     Building wheels for collected packages: node2vec
       Building wheel for node2vec (setup.py) ... done
       Created wheel for node2vec: filename=node2vec-0.4.3-py3-none-any.whl size=5980 sha256=65c5341a465a6cdbb89a6c4358e392195014a8685a97de03f0f6e378ae3cc1ca
       Stored in directory: /root/.cache/pip/wheels/07/62/78/5202cb8c03cbf1593b48a8a442fca8ceec2a8c80e22318bae9
     Successfully built node2vec
     Installing collected packages: node2vec
     Successfully installed node2vec-0.4.3
```

We now train the node2vec model on our graph (G_data):



```
[21] 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, 4618.72it/s]
     Generating walks (CPU: 1): 100%|██████████| 50/50 [00:26<00:00,  1.89it/s]
```

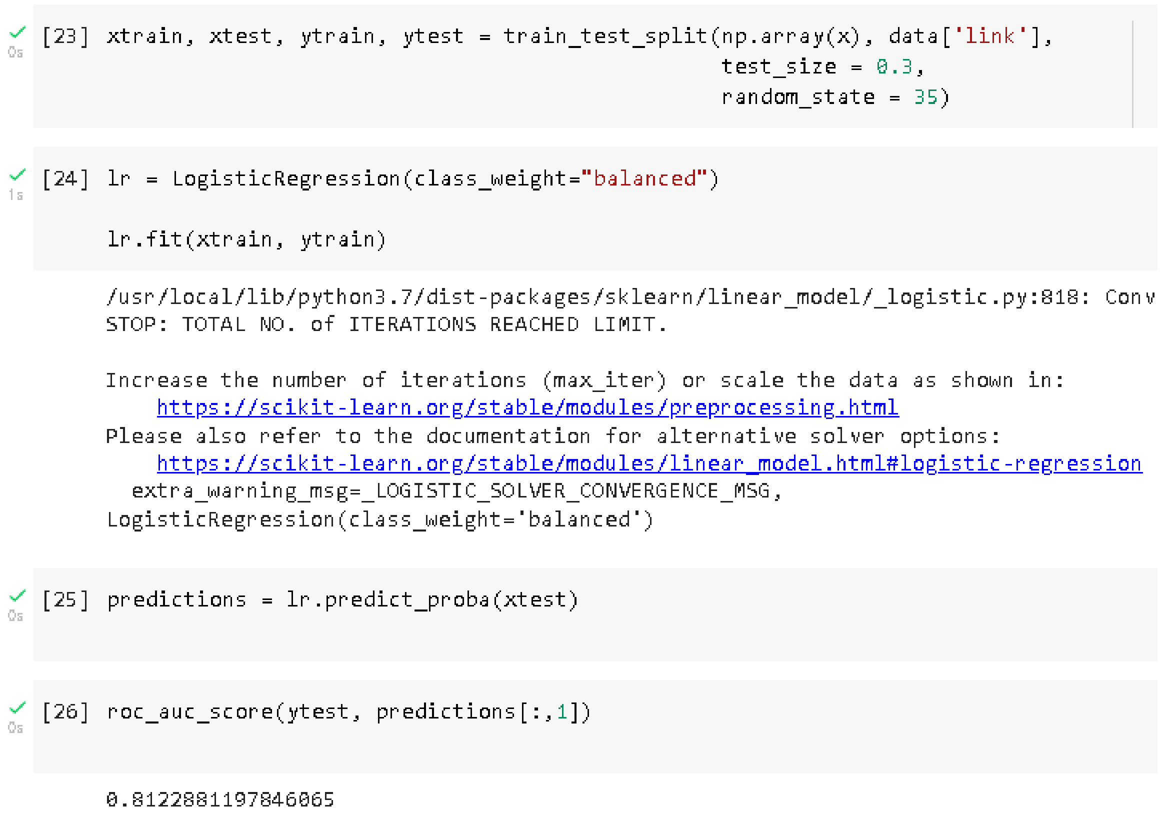
Apply the trained node2vec model to each unique pair of nodes in the data frame. To compute features for a pair or edge, add features for the nodes in that pair.



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

Building our Link Prediction Model

To validate the performance of the model, we need to split the data into two parts. One for training the model and one for testing the performance of the model**.**



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

```
[24] lr = LogisticRegression(class_weight="balanced")

     lr.fit(xtrain, ytrain)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: Conv
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
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
LogisticRegression(class_weight='balanced')
```

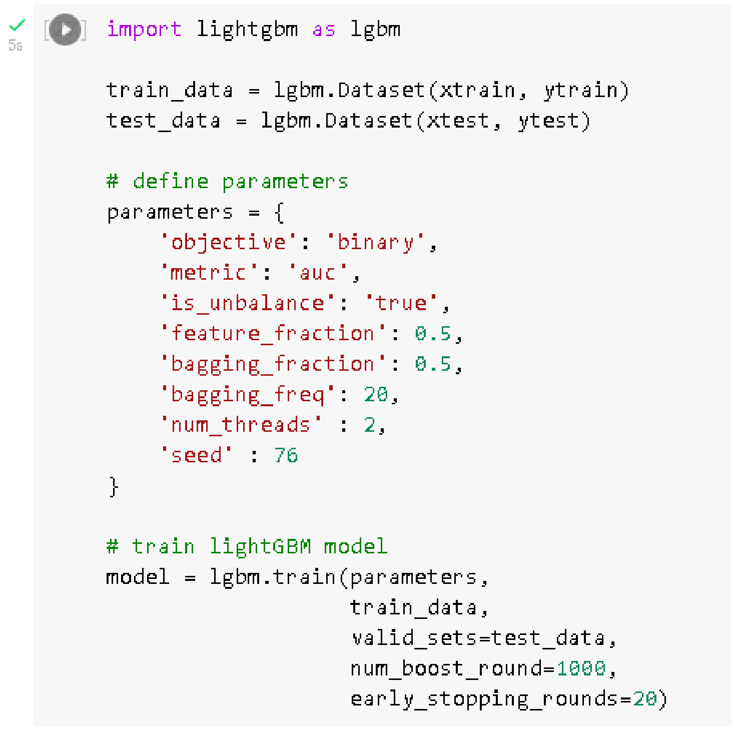
```
[25] predictions = lr.predict_proba(xtest)
```

```
[26] roc_auc_score(ytest, predictions[:,1])
```

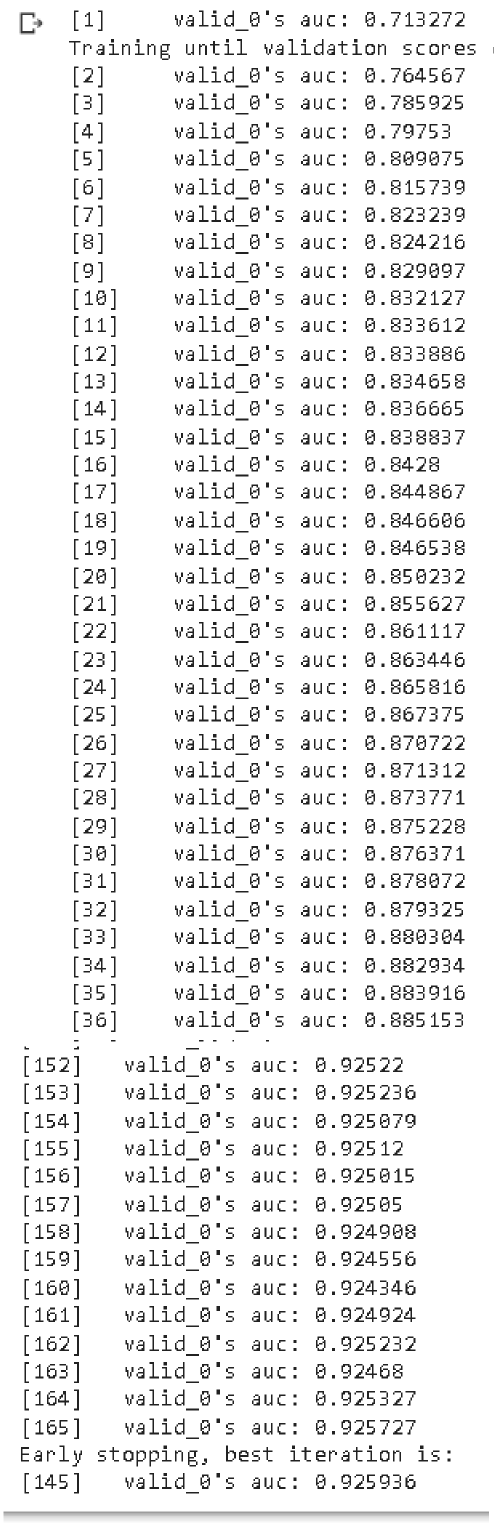
```
0.8122881197846065
```

To validate the performance of our model, we need to split the data into two parts. One to train the model and one to test the performance of the model:

- First, let's fit a logistic regression model:
- Score and check the performance of the model.
- Using the logistic regression model to get a score of 0.81.
- See if we can get a better score with a more complex model.



```
import lightgbm as lgbm

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)
```



```
[1]     valid_0's auc: 0.713272
Training until validation scores don't improve for 20 rounds.
[2]     valid_0's auc: 0.764567
[3]     valid_0's auc: 0.785925
[4]     valid_0's auc: 0.79753
[5]     valid_0's auc: 0.809075
[6]     valid_0's auc: 0.815739
[7]     valid_0's auc: 0.823239
[8]     valid_0's auc: 0.824216
[9]     valid_0's auc: 0.829097
[10]    valid_0's auc: 0.832127
[11]    valid_0's auc: 0.833612
[12]    valid_0's auc: 0.833886
[13]    valid_0's auc: 0.834658
[14]    valid_0's auc: 0.836665
[15]    valid_0's auc: 0.838837
[16]    valid_0's auc: 0.8428
[17]    valid_0's auc: 0.844867
[18]    valid_0's auc: 0.846606
[19]    valid_0's auc: 0.846538
[20]    valid_0's auc: 0.850232
[21]    valid_0's auc: 0.855627
[22]    valid_0's auc: 0.861117
[23]    valid_0's auc: 0.863446
[24]    valid_0's auc: 0.865816
[25]    valid_0's auc: 0.867375
[26]    valid_0's auc: 0.870722
[27]    valid_0's auc: 0.871312
[28]    valid_0's auc: 0.873771
[29]    valid_0's auc: 0.875228
[30]    valid_0's auc: 0.876371
[31]    valid_0's auc: 0.878072
[32]    valid_0's auc: 0.879325
[33]    valid_0's auc: 0.880304
[34]    valid_0's auc: 0.882934
[35]    valid_0's auc: 0.883916
[36]    valid_0's auc: 0.885153
[152]   valid_0's auc: 0.92522
[153]   valid_0's auc: 0.925236
[154]   valid_0's auc: 0.925079
[155]   valid_0's auc: 0.92512
[156]   valid_0's auc: 0.925015
[157]   valid_0's auc: 0.92505
[158]   valid_0's auc: 0.924908
[159]   valid_0's auc: 0.924556
[160]   valid_0's auc: 0.924346
[161]   valid_0's auc: 0.924924
[162]   valid_0's auc: 0.925232
[163]   valid_0's auc: 0.92468
[164]   valid_0's auc: 0.925327
[165]   valid_0's auc: 0.925727
Early stopping, best iteration is:
[145]   valid_0's auc: 0.925936
```

Training stopped after the 145th iteration because we applied the early stopping criterion. Most importantly, the model recorded an impressive AUC of 0.925936 on the test device.

4.Dataset Description:

We will work with a graph dataset in which the nodes are Facebook pages of popular food joints and well-renowned chefs from across the globe. If any two pages (nodes) like each other, then there is an edge (link) between them.

The dataset is prepared 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)

Link: https://networkrepository.com/fb-pages-food.php

5.Tools Used:

- Pandas: pandas are a software library written for the Python programming language for data manipulation and analysis. It offers data structures and operations for manipulating numerical tables and time series.
- NumPy: NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.
- Networkx: NetworkX is a Python library for studying graphs and networks
- tqdm: tqdm is a library in Python which is used for creating Progress Meters or Progress Bars
- re: A regular expression (or RE) specifies a set of strings that matches it; the functions in this module let you check if a string matches a given regular expression (or if a given regular expression matches a particular string, which comes down to the same thing).
- Matplotlib: Plotting library for the Python programming language and its numerical mathematics extension NumPy.
- Scikit-learn is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support-vector machines.
- node2vec- node2vec is an algorithmic framework for representational learning on graphs. Given any graph, it can learn continuous feature representations for the nodes, which can then be used for various downstream machine learning tasks.

6.Challenges Faced

a. Finding the right Project title
Identifying the right topics for your team can be tedious. To make sure projects are progressing according to plan and provides visibility into all use cases.

b. Communication issues.
During the initial days we were facing communication issues between our teams. Then we decided to be interactive and open between ourselves and be open to everyone so that team members can discuss issues with each other and work on overcoming it.

c. Lack of proper knowledge about software:
We had to research and learn about a lot of social economic networks topics and understand about it in detail in order to implement the project.

7.Contribution

Name (Roll number)	Contribution
Chirag Gupta (19z207)	Coding and model building
Darwin Debbarma(19z208)	Documentation and report writing
Varun Bhardwaj(19z256)	Coding and dataset preparation, Demo
Srinath S(20z433)	Research and topic selection
Vignesh A(20z436)	Research and topic selection
Rohit Sonar(19z342)	Documentation and dataset selection

References:

[1] P. Barford, A. Bestavros, A. Bradley and M. Crovella, Changes in web client access patterns: characteristics.

[2] [M.S. Chen, J.S. Park and P.S. Yu, Data mining for path traversal in a web environment, in: Proc. 16th International.

[3] D.W. Chueng, B. Kao and J.W. Lee, Discovering user access patterns on the World-Wide Web, in: Proc. First Pacific–Asia.

[4] W. Feller, Introduction to Probability Theory and Its Applications, Vols. 1 and 2, Wiley, New York,

P [5]J. Garofalakis, P. Kappos and D. Mourloukos, Web site optimization using page popularity, IEEE Internet Computing,

[6] D. Gibson, J. Kleinberg and P. Raghavan, Inferring web communities from link topology, in: Proc. 9th ACM conference on.

[7] T. Joachims, D. Freitag and T. Mitchell, WebWatcher: a tour guide for the World Wide Web.
[8] J. Kleinberg, Authoritative sources in a hyperlinked environment, in: Proc. 9th ACM-SIAM Symposium on Discrete
.
[9] A. Kraiss and G. Weikum, Integrated document caching and prefetching in storage hierarchies based on Markov-chain.