Social Media Analytics #HW1

Link Prediction

M11007601 周沂潔

一、說明如何執行程式(並附上程式碼檔案)

程式撰寫在 **Google Colab** 中，

匯入 (1) data_train_edge.csv (2) predict.csv (3) ans500_ground_truth.csv 後，

可以直接執行，結果會顯示出 predict accuracy，confusion matrix 以及匯出一個 result file。

二、簡介你所使用的程式架構及演算法流程(如果有進行前處理也請解釋原因)

A. Strategy to Solve the Link Prediction Problem

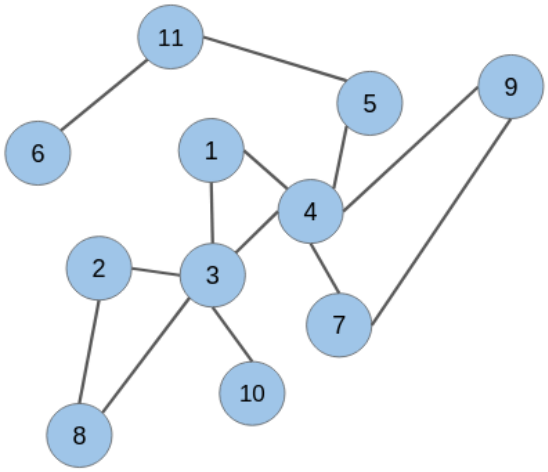

Fig.1 Original Graph

- 原本的 Graph 的連線狀態，在不影響 Graph 的 connected component 數量及原本 node 數量情況下，將幾條 link 拔掉，作為 training 的 data。

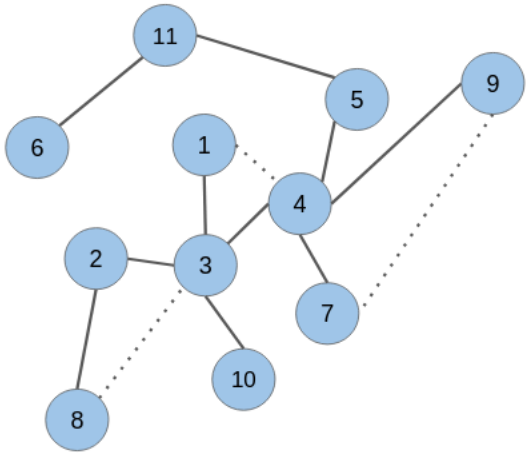

Fig.2 Graph after remove some links

- 將移除的 link 與原本就沒有連線的 link 結合，並加上 label。
- 0 unconnected
- 1 connected

Features	Link (Target Variable)
Features of node pair 1 – 2	0
Features of node pair 1 – 5	0
Features of node pair 1 – 7	0
Features of node pair 1 – 8	0
Features of node pair 1 – 9	0
Features of node pair 1 – 10	0
Features of node pair 2 – 4	0
Features of node pair 2 – 10	0
Features of node pair 3 – 5	0
Features of node pair 3 – 7	0
Features of node pair 3 – 9	0
Features of node pair 4 – 8	0
Features of node pair 4 – 10	0
Features of node pair 4 – 11	0
Features of node pair 5 – 6	0
Features of node pair 5 – 7	0
Features of node pair 5 – 9	0
Features of node pair 8 – 10	0
Features of node pair 1 – 4	1
Features of node pair 3 – 8	1
Features of node pair 7 – 9	1

Fig.3 Features and Labels

會發現 label 的數量差異很大，而在實際情況也是 unconnected node pairs 會佔多數。

B. 程式架構及演算法流程

Import libraries

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

讀取資料集

```
dataset = pd.read_csv('data_train_edge.csv')
test_set = pd.read_csv('predict.csv')
x = dataset.iloc[:, 0].values
y = dataset.iloc[:, 1].values
ans500 = pd.read_csv('ans500_ground_truth.csv')
y_test = ans500.iloc[:,1]
```

將沒有連線到的點記錄下來

```
independent_nodes = []
for i in range(1005):
  if i not in x and i not in y:
    independent_nodes.append(i)
```

建立 adjacency list

```
matrix = np.zeros((1005,1005))

for i in range(len(x)):
  row = x[i]
  col = y[i]
  matrix[row][col] = 1
```

建立 Graph 以利後續計算

```
# create graph
G = nx.from_pandas_edgelist(dataset, "node1", "node2", create_using=nx.DiGraph())
independent_nodes_counts = len(independent_nodes)
for i in range(independent_nodes_counts):
  G.add_node(independent_nodes[i])
```

依 adjacency list 來找出所有 unconnected node pairs

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

# traverse adjacency matrix
for i in range(matrix.shape[0]):
  for j in range(matrix.shape[1]):
    if matrix[i,j] == 0:
      all_unconnected_pairs.append([i,j])

print(len(all_unconnected_pairs))
```

將所有 unconnected node pair 存入 data 中，並加上 label 0。

```
[ ]  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({'node1':node_1_unlinked,
                          'node2':node_2_unlinked})

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

取出所有不影響 Graph 中 number of connected components 和 initial node count 的 link。

```
initial_node_count = len(G.nodes)
initial_number_connected_components = nx.number_weakly_connected_components(G)
dataset_temp = dataset.copy()

# empty list to store removable links
omissible_links_index = []
# print(dataset_temp)

for i in tqdm(dataset.index.values):
  # remove a node pair and build a new graph
  test = dataset_temp.drop(index = i)
  G_temp = nx.from_pandas_edgelist(test, "node1", "node2", create_using=nx.DiGraph())
  for n in range(independent_nodes_counts):
    G_temp.add_node(independent_nodes[n])
  # check there is no spliting of graph and number of nodes is same
  if (nx.number_weakly_connected_components(G_temp) == initial_number_connected_components) and (len(G_temp.nodes) ==
    omissible_links_index.append(i)
    dataset_temp = dataset_temp.drop(index = i)
```

執行儲存格 (⌘/Ctrl+Enter)
尚未在這個工作階段中執行儲存格

將剛剛移除的 link 加入剛剛的 data。

```
[ ]  # create dataframe of removable edges
     dataset_removable = dataset.loc[omissible_links_index]

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

     data = data.append(dataset_removable[['node1', 'node2', 'link']], ignore_index=True)

     print(data['link'].value_counts())

     0    989568
     1     19479
     Name: link, dtype: int64
```

並建立移除後的新的 graph_new。

```
[ ]  # drop removable edges
     dataset_partial = dataset.drop(index=dataset_removable.index.values)

     # build graph
     G_data = nx.from_pandas_edgelist(dataset_partial, "node1", "node2", create_using=nx.Graph())
     for n in range(independent_nodes_counts):
         G_data.add_node(independent_nodes[n])
```

使用 node2vec 建立出 graph_new 的 feature。

- num_walks : 為 Graph 中每一個 node 來走多少 random walk 的次數
- walk_length : 為每一個 random walk 走的長度

```
from node2vec import Node2Vec

# Generate walks
node2vec = Node2Vec(G_data, dimensions=128, walk_length=200, num_walks=100)

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

```
Computing transition probabilities: 100% | 1005/1005 [00:00<00:00, 3958.81it/s]
Generating walks (CPU: 1): 100%| | 100/100 [12:23<00:00,  7.44s/it]
```

依 feature 建立出 training 和 testing 的資料集。

```
[ ] X_train = [(n2w_model[str(i)]+n2w_model[str(j)]) for i,j in zip(data['node1'], data['node2'])]
    y_train = data['link']
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: DeprecationWarning: Call to deprecated
  """Entry point for launching an IPython kernel.
```

```
[ ] X_test = [(n2w_model[str(i)]+n2w_model[str(j)]) for i,j in zip(test_set['node1'], test_set['node2'])]
```

做特徵縮放，將標準差調為 1 平均調為 0

```
[ ] # Feature Scaling
    from sklearn.preprocessing import StandardScaler
    sc = StandardScaler()
    X_train = sc.fit_transform(X_train)
    X_test = sc.transform(X_test)
```

使用 light gbm

```
[25] # light gbm
     import lightgbm as lgb
     from sklearn.metrics import accuracy_score
     lgbclf = lgb.LGBMClassifier(max_depth=9, num_leaves=511, learning_rate=0.04, n_estimators=1000, early_stop_rou
     lgbclf.fit(X_train,y_train)
     lgb_predictions = lgbclf.predict(X_test)
     print(len(lgb_predictions))
     print("Accuracy:", accuracy_score(y_test, lgb_predictions[0:500]))
```

- learning_rate : 學習速率，選擇小一點的學習速率會獲得較穩定的模型
- n_estimators : boosting 的迭代次數，選擇較大的值配合 early_stopping_round 來讓模型自動選擇最好的迭代次數
- mean_child_sample : 根據資料量來決定，資料量大時提昇讓葉子結點分不穩定
- max_depth : 模型的最大深度
- num_leaves : 一顆樹上的葉子節點，一般設置為 < 2 ^ max_depth - 1

顯示出 Confusion Matrix，查看前五百筆資料預測結果。

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, lgb_predictions[0:500])
print('Confusion matrix\n\n', cm)
print('\nTrue Positives(TP) = ', cm[0,0])
print('\nTrue Negatives(TN) = ', cm[1,1])
print('\nFalse Positives(FP) = ', cm[0,1])
print('\nFalse Negatives(FN) = ', cm[1,0])
```

```
Confusion matrix

 [[239   7]
 [ 28 226]]

True Positives(TP) =  239

True Negatives(TN) =  226

False Positives(FP) =  7

False Negatives(FN) =  28
```

匯出預測結果，會在資料夾中產生新的 csv 檔。

```
[ ] import csv
    with open('lgb_predict_result13_0.93.csv', 'w', newline='') as csvfile:
      # 建立 CSV 檔寫入器
      writer = csv.writer(csvfile)

      # 寫入一列資料
      writer.writerow(['predict_nodepair_id', 'ans'])
      for i in range(len(lgb_predictions)):
        writer.writerow([i, lgb_predictions[i]])
```