Social Media Analytics #HW2

Community Detection

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) train.csv (2) test.csv 後，

可以直接執行，匯出一個 community_detection.csv 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 Community Detection

使用 Community-louvain 的 Library 來實作。

- Community.best_partition(graph, partition=None, weight='weight', resolution=1.0)
  是使用 Louvain Heurictices 方法劃分的獲得最高 modularity 的 community detection 演算法
- Louvain Algorithm
  Community 內的 edge weight 和 Community 之間的 edge weight 的比例，edge weight 越大則 modularity 越大，community 內的 node 更緊密，劃分的質量越好。

$$Q = \frac{1}{2m}\sum_{i,j}\left[A_{ij} - \frac{k_i k_j}{2m}\right]\delta(c_i, c_j)$$

M 為 Graph 的 edge 數量，2m 為總 degree 數。

A 為兩點 i j 相連時為 1，否為 0。

$\delta(c_i, c_j)$ 為 i j 在同一 community 時為 1，否為 0。

- Louvain Flow Chart

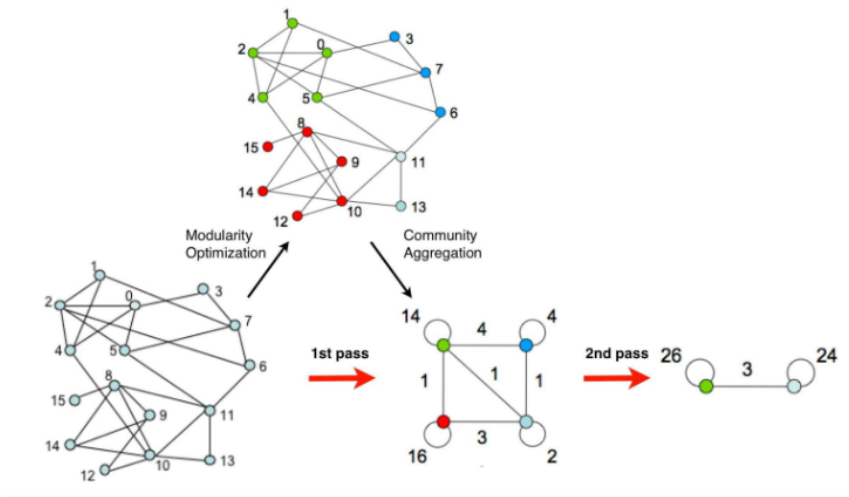

Fig 1. Louvain Flow Chart

(1) 每個 node 都先分配一個單獨的 community，再去嘗試與自己連接的節點歸入對方的 community，計算加入不同鄰居 community 之後的 modularity gain，最後將自己歸到 modularity gain 最大的 community 中。

$$\Delta Q = \left[\frac{\sum_{in}+k_{i,in}}{2m} - \left(\frac{\sum_{tot}+k_i}{2m}\right)^2\right] - \left[\frac{\sum_{in}}{2m} - \left(\frac{\sum_{tot}}{2m}\right)^2 - \left(\frac{k_i}{2m}\right)^2\right]$$

(2) 將每個 community 變成 super node，supernode 與 supernode 之間的 weight 就是兩個 community 之間的 edge 數量。
supernode 自己的 weight 就是 community 內 degree 的數量

(3) 回到 (1) 不斷的迭代，每次網路中的初始 community 數量會變小，因此迭代的速度會更快。

B. 程式架構及演算法流程

Import libraries

```
import community.community_louvain
import matplotlib.cm as cm
import matplotlib.pyplot as plt
import networkx as nx
import pandas as pd
from tqdm import tqdm
import numpy as np
import csv
```

讀取測試資料集

```
train_set = pd.read_csv('train.csv')
trainNode1 = train_set.iloc[:, 0].values
trainNode2 = train_set.iloc[:, 1].values
nodeCount = max(max(trainNode1),max(trainNode2))
```

將沒有連線到的點記錄下來

```
independentNode = []
for i in tqdm(range(nodeCount+1)):
  if i not in trainNode1 and i not in trainNode2:
    independentNode.append(i)
```

將沒有連線到的 node 與最大值的 node+10 連在一起（與其他 community 區隔開來）

```
independentNodeCount = len(independentNode)
for i in tqdm(range(independentNodeCount)):
  trainNode1 = np.append(trainNode1, independentNode[i])
  trainNode2 = np.append(trainNode2, nodeCount+10)
```

將連線好的 independentNode 和原本的 node append 後做 sorting 後會出 trainEdge.csv。

```
list1, list2 = zip(*sorted(zip(trainNode1, trainNode2)))
```

```
with open('trainEdge.csv', 'w', newline='') as csvfile:
  # 建立 CSV 檔寫入器
  writer = csv.writer(csvfile)

  # 寫入一列資料
  writer.writerow(['Node1','Node2'])
  for i in range(len(list1)):
    writer.writerow([list1[i],list2[i]])
```

使用 networkx 建立 graph，並用 community_louvain.best_partition 做 detection。

```
with open('trainEdge.csv', 'w', newline='') as csvfile:
  # 建立 CSV 檔寫入器
  writer = csv.writer(csvfile)

  # 寫入一列資料
  writer.writerow(['Node1','Node2'])
  for i in range(len(list1)):
    writer.writerow([list1[i],list2[i]])
```

最後以 test.csv 匯出預測結果

```
test_set = pd.read_csv('test.csv')
testNode1 = test_set.iloc[:, 1].values
testNode2 = test_set.iloc[:, 2].values
testLength = len(testNode1)

prediction = []
for i in range(testLength):
    if partition[testNode1[i]] == partition[testNode2[i]]:
      prediction.append('1')
    else:
      prediction.append('0')
```

```
with open('community_detection.csv', 'w', newline='') as csvfile:
  # 建立 CSV 檔寫入器
  writer = csv.writer(csvfile)

  # 寫入一列資料
  writer.writerow(['Id','Category'])
  for i in range(len(prediction)):
    writer.writerow([i,prediction[i]])
```