Practice of Social Media Analytics

Final Project：

Link Prediction on different datasets

Contents

1. Introduction

我們先使用 networkx 這個 library 來對每個 Dataset 建立 Graph，然後使用上課所教的 Link Prediction 方法(common neighbors, Jaccard coefficient, preferential attachment, Adamic/Adar)，先去計算每個 Dataset 裡面，每條已經存在的 edge 的分數。

然後根據把每種分數做 normalize 後加總起來，接著再把最高分數的一些 edge 移除。

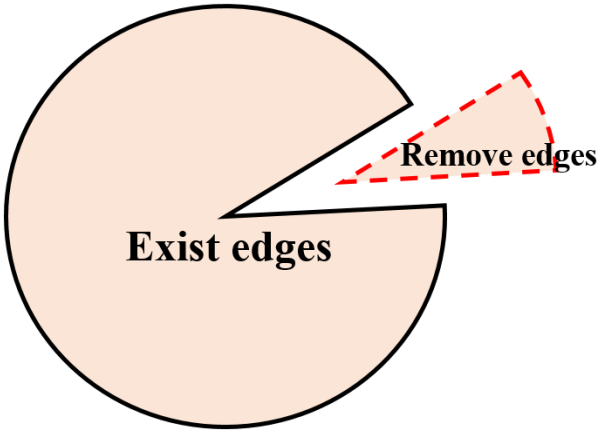
預測的結果則是去算移除部分 edge 的 Graph 裡面，所有 node 之間的分數，再由投票進行 edge 的預測，最後比較投票出來的 edge 跟所移除 edge 的之間的準確率。

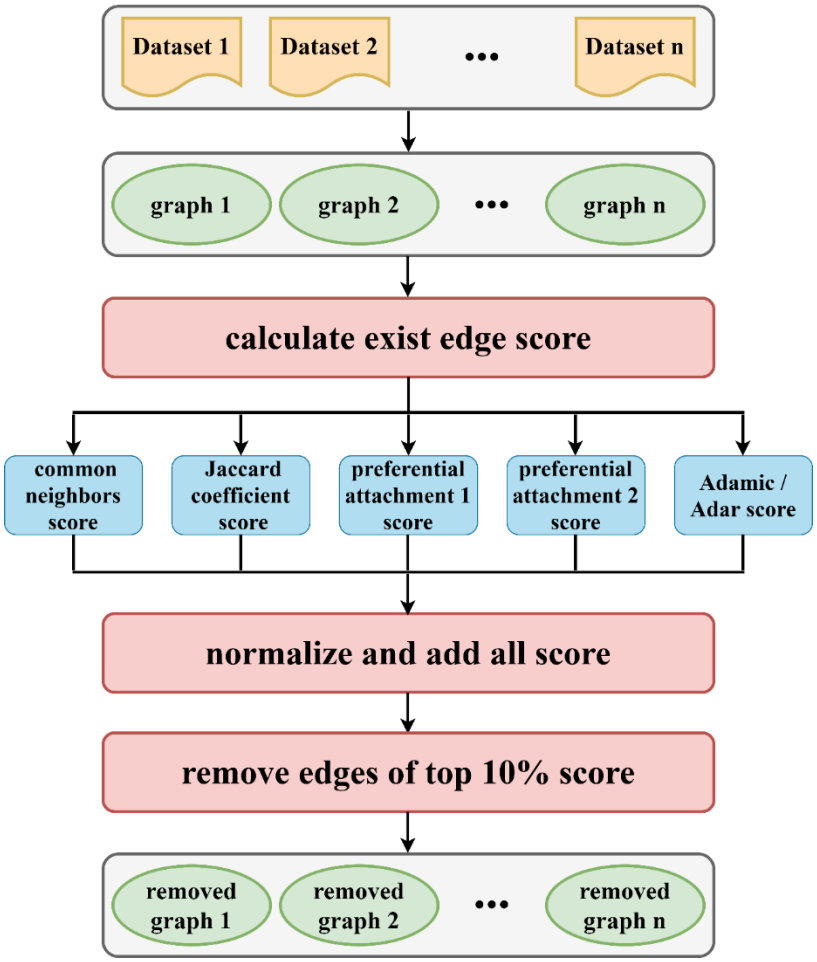
我們實驗會對不同的 dataset 進行測試，藉由觀察到不同 dataset 的 degree distribution，來推測預測結果不同之原因。

2. Method

2-1. Remove Edge Method

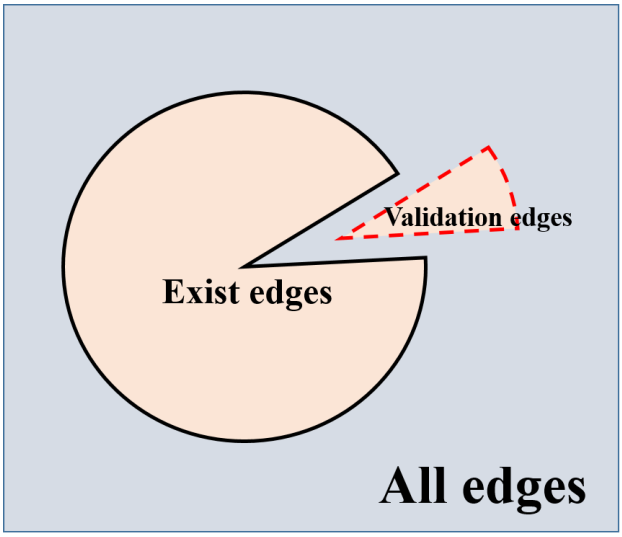
我們移除 edge 的方法，主要是先計算每條 exist edge 的 5 種 Link Prediction 分數，然後再對這些分數做 normalize 接著全部加起來，最後取最高的前幾 percent 來做移除。

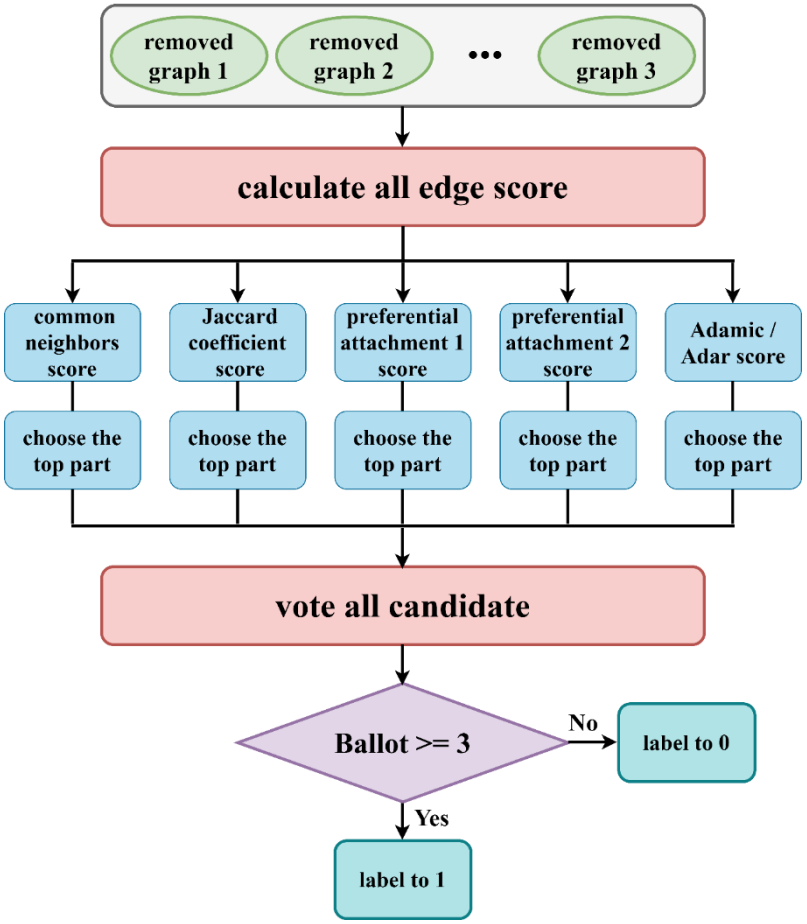




2-2. Predict Edge Method

我們預測 edge 的方法，主要是先對被移除完 edge 的 Graph，計算所有節點之間的 5 種 Link Prediction 分數，然後再對每種分數取前幾名出來，接著再做投票，如果票數超過 3 票，就預測為有，最後再計算準確率。





2-3. Code Explanation

執行環境：使用 Kaggle 上的 Notebook 撰寫程式與執行。

2-3-1. 使用 numpy、pandas 基本套件，以及 networkx 來建立 Dataset 的 Graph。

```
import numpy as np
import pandas as pd
import networkx as nx

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

2-3-2. 讀取 Dataset，然後建立 node 和 edge。

```
edge_df=pd.read_csv('/kaggle/input/proximity-network/proximity_network.csv')

G = nx.Graph()
G.add_nodes_from(edge_df.node1)
G.add_nodes_from(edge_df.node2)

node1_list=edge_df.node1.tolist()
node2_list=edge_df.node2.tolist()
node_number=list(set(node1_list)|set(node2_list))

edge_list=[]
for i in range(len(node1_list)):
    node1=node1_list[i]
    node2=node2_list[i]

    edge_list.append([node1,node2])

for i in range(len(node1_list)):
    G.add_edge(node1_list[i],node2_list[i])
```

2-3-3. 五種 Link Prediction 的方法，其中 preferential attachment 用相加和相乘兩種方法。

```
"""CN"""
def common_neighbors_score(G,u,v):
    return len(set(G.neighbors(u)) & set(G.neighbors(v)))

def common_neighbors(G,node1_list,node2_list,output_list):

    for i in range(len(node1_list)):
        node1=node1_list[i]
        node2=node2_list[i]
        if (node1 in node_number) and (node2 in node_number):
            length=common_neighbors_score(G,node1,node2)
            output_list.append(length)
        else:
            output_list.append(0)

"""JC"""
def Jaccard_coeffieient_score(G,u,v):
    intersection_length=len(set(G.neighbors(u)) & set(G.neighbors(v)))
    union_length=len(set(G.neighbors(u)) | set(G.neighbors(v)))
    if intersection_length is 0:
        return 0
    else:
         return float(intersection_length/union_length)

def Jaccard_coeffieient(G,node1_list,node2_list,output_list):

    for i in range(len(node1_list)):
        node1=node1_list[i]
        node2=node2_list[i]
        if (node1 in node_number) and (node2 in node_number):
            length=Jaccard_coeffieient_score(G,node1,node2)
            output_list.append(length)
        else:
            output_list.append(0)

"""PA1"""
def preferential_attachment_score1(G,u,v):
    return G.degree(u)*G.degree(v)
```

2-3-4. 計算五種每條已存在 edge 的分數，然後做 normalize。

```
CN,JC,PA1,PA2,Ad=[],[],[],[],[]
column_name=['CN','JC','PA1','PA2','Ad']

common_neighbors(G,node1_list,node2_list,CN)
Jaccard_coeffieient(G,node1_list,node2_list,JC)
preferential_attachment_1(G,node1_list,node2_list,PA1)
preferential_attachment_2(G,node1_list,node2_list,PA2)
Adamic(G,node1_list,node2_list,Ad)

df_t=pd.DataFrame(list(zip(CN,JC,PA1,PA2,Ad)),columns=column_name)
n_df_t=(df_t-df_t.min())/(df_t.max()-df_t.min())
df_t=n_df_t
df_t
```

2-3-5. 接著把五種分數相加。

```
score=[]
for i in range(len(df_t.CN)):
    sum_row=\
    df_t.CN[i] +\
    df_t.JC[i] +\
    df_t.PA1[i] +\
    df_t.PA2[i] +\
    df_t.Ad[i]

    score.append(sum_row)
```

2-3-6. 定義把最高分數的前幾條 edge 拿出來

```
def label_top(column,label,edge_count):
    score=[]
    if type(column) is not type(score):
        score=column.tolist()
    else:
        score=column

    idx_list = sorted(range(len(score)), key = lambda k: score[k])

    for i in range(len(score)):
        label.append(1)

    for i in range(len(score)-edge_count):
        index=idx_list[i]
        label[index]=0
```

2-3-7. 移除 1%、5%、10%的 edge (此範例移除 1%的 edge)，然後創建移除 edge 後的 Graph，再把移除的 edge 記錄下來。

```
remove_edges_count=round(G.number_of_edges()*0.01)
label=[]
label_top(score,label,remove_edges_count)

G_new=G.copy()
remove_list=[]
for i in range(len(label)):
    if label[i] == 1:
        u=node1_list[i]
        v=node2_list[i]
        G_new.remove_edge(u,v)
        remove_list.append([u,v])
```

2-3-8. 把所有需要 predict 的 edge 給記錄起來。

```
predict_edge=[]
exist_edge=[]
for u in range(len(list(G_new.nodes))):
    for v in range(u+1,len(list(G_new.nodes))+1):
        if G_new.has_edge(u, v):
            exist_edge.append([u,v])
        else:
            predict_edge.append([u,v])

predict_node1=[]
predict_node2=[]
for i in range(len(predict_edge)):
    node1=predict_edge[i][0]
    node2=predict_edge[i][1]
    predict_node1.append(node1)
    predict_node2.append(node2)
```

2-3-9. 計算所有 predict edge 的五種 Link Prediction 的分數。

```
CN,JC,PA1,PA2,Ad=[],[],[],[],[]
column_name=['CN','JC','PA1','PA2','Ad']

common_neighbors(G_new,predict_node1,predict_node2,CN)
Jaccard_coeffieient(G_new,predict_node1,predict_node2,JC)
preferential_attachment_1(G_new,predict_node1,predict_node2,PA1)
preferential_attachment_2(G_new,predict_node1,predict_node2,PA2)
Adamic(G_new,predict_node1,predict_node2,Ad)


df_p=pd.DataFrame(list(zip(CN,JC,PA1,PA2,Ad)),columns=column_name)
n_df_p=(df_p-df_p.min())/(df_p.max()-df_p.min())
df_p=n_df_p
df_p
```

2-3-10. 把每個五種分數前幾名的 edge 記錄起來。

```
CN,JC,PA1,PA2,Ad=[],[],[],[],[]
coe=3

label_top(df_p.CN,CN,remove_edges_count*coe)
label_top(df_p.JC,JC,remove_edges_count*coe)
label_top(df_p.PA1,PA1,remove_edges_count*coe)
label_top(df_p.PA2,PA2,remove_edges_count*coe)
label_top(df_p.Ad,Ad,remove_edges_count*coe)
```

2.3.11. 對所有 edge 做投票，只要有 edge 超過三票就預測為有可能的 edge。

```
vote=[]
for i in range(len(CN)):
    vote.append(CN[i]+JC[i]+Ad[i]+PA1[i]+PA2[i])

vote_number=len(column_name)
score=[]
for i in range(len(vote)):
    if vote[i] >= vote_number:
        score.append(vote_number)
    elif vote[i] >= (vote_number-1):
        score.append(vote_number-1)
    elif vote[i] >= (vote_number-2):
        score.append(vote_number-2)
    else:
        score.append(0)
```

2-3-11. 把多餘的 edge 給過濾掉，只留下要預測的數量。

```
label=[]
label_top(score,label,remove_edges_count)
```

2.3.12. 把剛預測的 edge 轉換成 list，因為無向圖，所以用兩個 list。

```
top_list1=[]
top_list2=[]
for i in range(len(label)):
    if label[i] != 0:
        top_list1.append([predict_node1[i],predict_node2[i]])
        top_list2.append([predict_node2[i],predict_node1[i]])
```

2-3-12. 最後計算準確率。

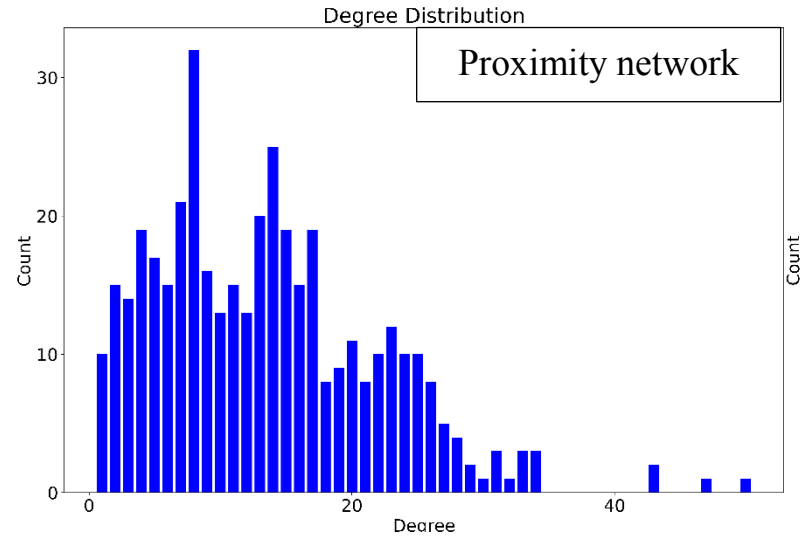
```
cnt=0
for i in range(len(top_list1)):
    if top_list1[i] in remove_list or top_list2[i] in remove_list:
        cnt+=1

cnt/remove_edges_count
```

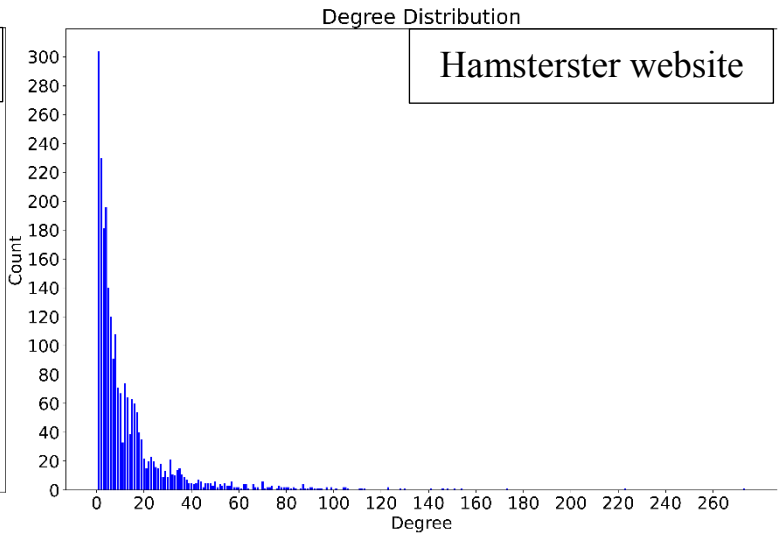
3. Experiment

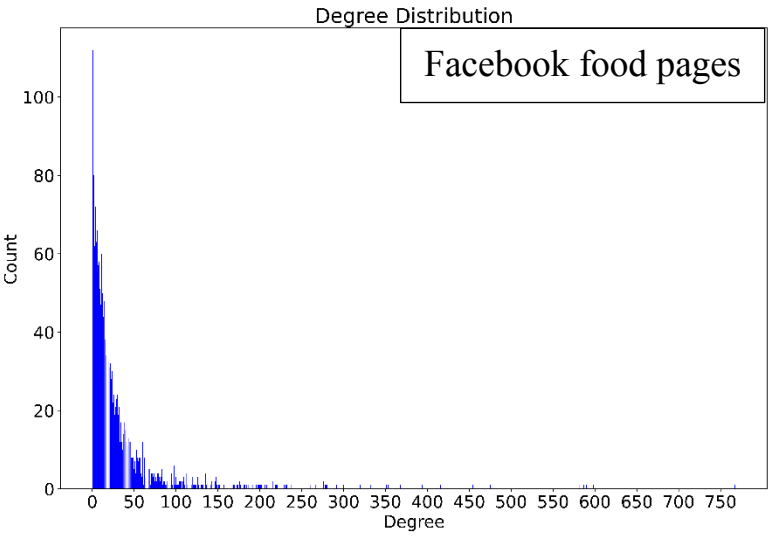
3-1. Dataset Characteristic

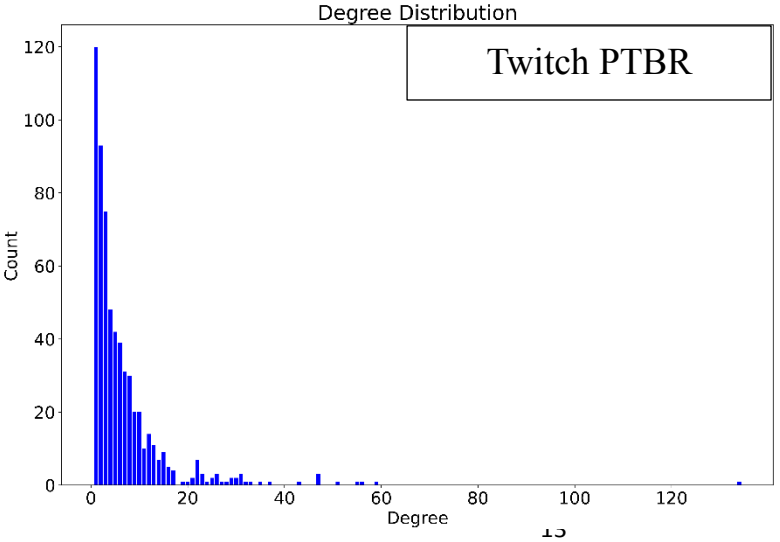
Dataset	# nodes	# edges	description
Proximity network	410	2765	A human contact network.
Hamsterster website	2426	16630	The friendships and family links between users of the website.
Twitch PTBR	1912	31299	Twitch user-user networks of gamers in a certain language.
Facebook food pages	620	2102	Facebook page networks of food.

下圖是四個 Dataset 的 Degree Distribution，可以看得出來只有 Proximity network 比較沒有這麼符合 power law。



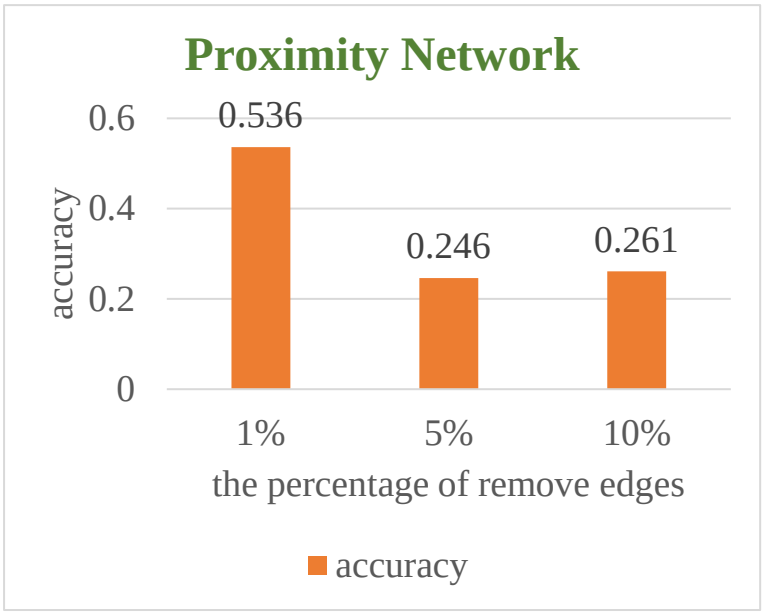


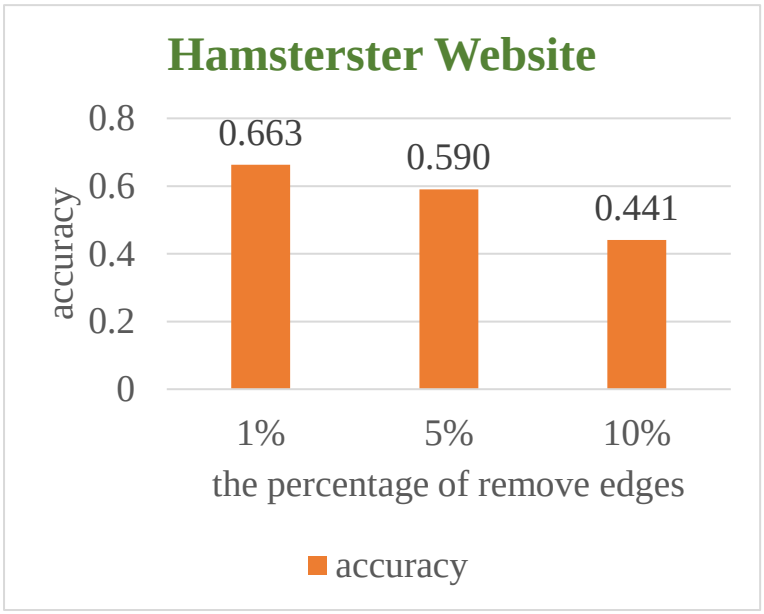


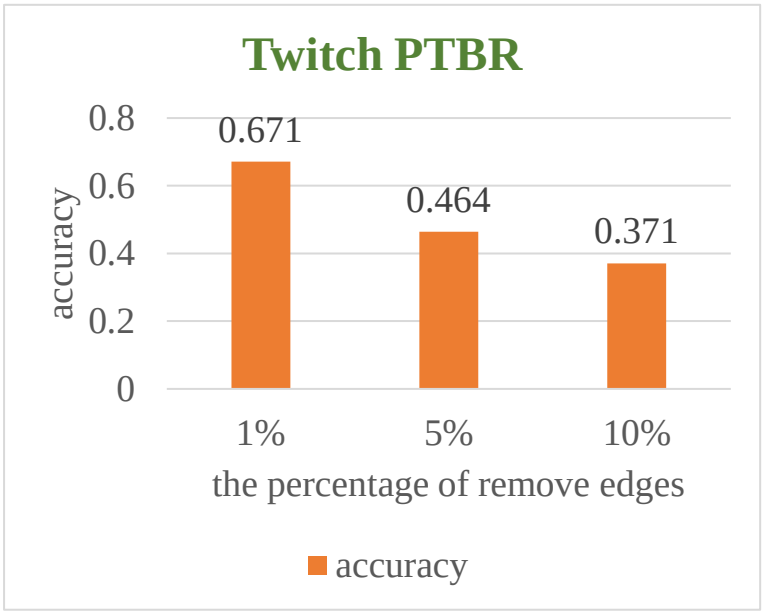


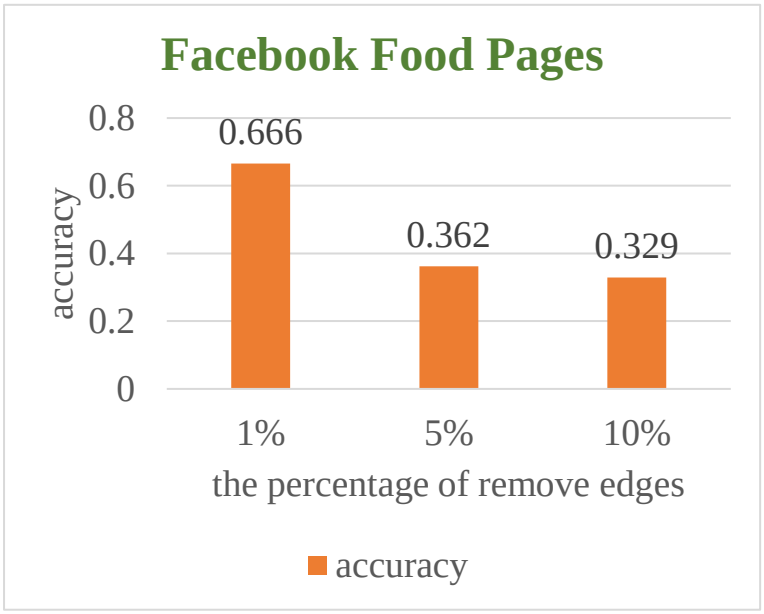
3-2. Evaluation

我們主要是測試把 1%、5%、10%的 edge 拿掉之後的準確率，可以很直覺的看出拿掉 1%的 edge 因為損失的訊息量較少，所以準確率會比較高，其中 Proximity Network 的準確率是四個之中為低的，可能是它的 Degree Distribution 比較不符合 power law 所以預測的 edge 會包含比較多不是本來的 edge。









4. Conclusion

我們使用五種 Link Prediction 的方法，先對分數做 normalize 然後加總，之後對原本資料集的 edge 做移除。

預測的方法則是先算所有可能 edge 的分數，最後再用投票的方法做驗證。

我們實驗主要使用四種資料集，計算移除 1%、5%、10%的 edge 準確率，發現在 Proximity Network 這個資料集的準確率比其他三種還要低一些，可能是因為它比較沒有符合 power law 定律。

最後我們總結出拿掉 1%的 edge 的準確率是都會大於 50%，主要還是因為整體的訊息量損失的較少。