DL2024 期末專案

Final Presentation

第二組

資訊所 賴瑜萱

資訊所 莊上緣

數據所 李易庭

敏求機器人 詹雅淇

Outline

- Problem Definition
- Dataset
- Pipeline
- Method
- Experiments
- Future Works & Conclusion

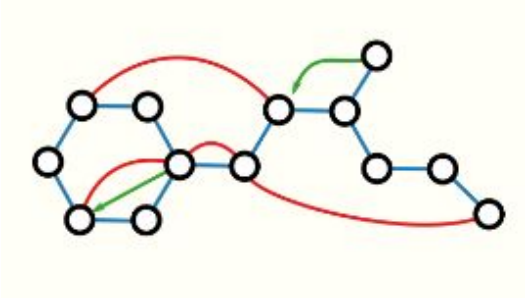
Outline

- **Problem Definition**
- Dataset
- Pipeline
- Method
- Experiments
- Future Work & Conclusion

Problem Definition

在**label嚴重不平衡**，且特徵幾乎都經過**匿名處理**的信用卡交易紀錄表格式資料集實現高準確度的**GNN圖神經網路異常檢測模型**

GNN圖神經網路



TDL表格式資料學習

異常檢測

Outline

- Problem Definition
- **Dataset**
- Pipeline
- Method
- Experiments
- Future Work & Conclusion

Dataset

[Credit Card Fraud Detection (kaggle.com)]

```
df=pd.read_csv('creditcard.csv')
print(df.shape)
print(df.columns)
```

```
(284807, 31)
Index(['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10',
       'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20',
       'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount',
       'Class'],
      dtype='object')
```

這是一個類別非常不平衡的資料集，label被標記為詐騙的資料，在28萬筆中只有不到500筆，只佔了所有資料的0.172%

```
[27]: print("Class:\n")
      print(df['Class'].describe())
      print(df['Class'].value_counts())
```

```
Class:

count    284807.000000
mean          0.001727
std           0.041527
min           0.000000
25%           0.000000
50%           0.000000
75%           0.000000
max           1.000000
Name: Class, dtype: float64
Class
0    284315
1       492
Name: count, dtype: int64
```

creditcard.csv (150.83 MB)

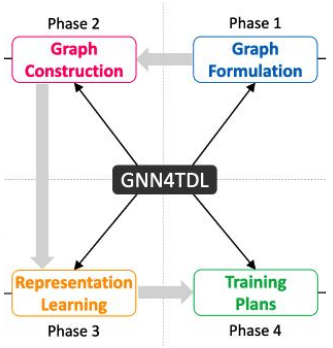
Detail | Compact | Column | 10 of 31 columns

# Time Number of seconds elapsed between this transaction and the first transaction in the dataset	# V1 may be result of a PCA Dimensionality reduction to protect user identities and sensitive features(v1-v28)	# V2	# V3	# V4	# V5
0 – 173k	-56.4 – 2.45	-72.7 – 22.1	-48.3 – 9.38	-5.68 – 16.9	-114
0	-1.3598071336738	-0.0727811733098497	2.53634673796914	1.37815522427443	-0.33832
0	1.19185711131486	0.26615071205963	0.16648011335321	0.448154078460911	0.060017
1	-1.35835406159823	-1.34016307473609	1.77320934263119	0.379779593034328	-0.50319
1	-0.966271711572087	-0.185226008082898	1.79299333957872	-0.863291275036453	-0.01030
2	-1.15823309349523	0.877736754848451	1.548717846511	0.403033933955121	-0.40719
2	-0.425965884412454	0.960523044882985	1.14110934232219	-0.168252079760302	0.420986
4	1.22965763450793	0.141003507049326	0.0453707735899449	1.20261273673594	0.191880
7	-0.644269442348146	1.41796354547385	1.0743803763556	-0.492199018495015	0.948934
7	-0.89428608220282	0.286157196276544	-0.113192212729871	-0.271526130088604	2.669598

Outline

- Problem Definition
- Dataset
- **Pipeline**
- Method
- Experiments
- Future Work & Conclusion

pipeline



creditcard.csv

stage1:
graph formulation

stage2:
graph construction

stage3:
分成 ego 和 neighbor

stage4:
計算 anomaly scores

Ego features

Anonymized message passing

Neighbor features

Pre-processing module

Linear

Linear

Cosine Sim

s_1: 0.11
s_2: 0.05
s_3: 0.14
s_4: 0.73
s_5: 0.21
s_6: 0.10
s_7: 0.09

Anomaly scores

Ego-neighbor matching module

Outline

- Problem Definition
- Dataset
- Pipeline
- **Method**
- Experiments
- Future Work & Conclusion

0. Preprocessing

Sampling
-> benign node:fraud node=10:1

```python
# 10:1的資料
def data_sampling(df):
    # print(df)
    fraud_data=df[df['Class']==1]
    normal_data=df[df['Class']==0].sample(n=len(fraud_data)*10,random_state=42)
    # print(fraud_data.shape)
    # print(normal_data.shape)
    selected_df=pd.concat([normal_data,fraud_data])
    selected_df.reset_index(drop=True,inplace=True)
    return selected_df
```

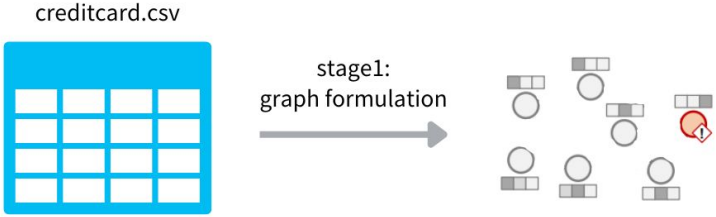
```python
df=data_sampling(df)
print(df.index)
```

```
RangeIndex(start=0, stop=5412, step=1)
```

Standardization
-> ’Time’ , ’Amount’

```python
# 標準化
def data_standardization(df):
    # 假設 df 是你的 DataFrame，columns 是要標準化的欄位名稱列表
    columns_to_normalize = ['Time', 'Amount']  # 填入要標準化的欄位名稱
    # 初始化 StandardScaler 物件
    scaler = StandardScaler()
    # 對指定列進行標準化
    df[columns_to_normalize] = scaler.fit_transform(df[columns_to_normalize])
    return df
```

1.Graph Formulation

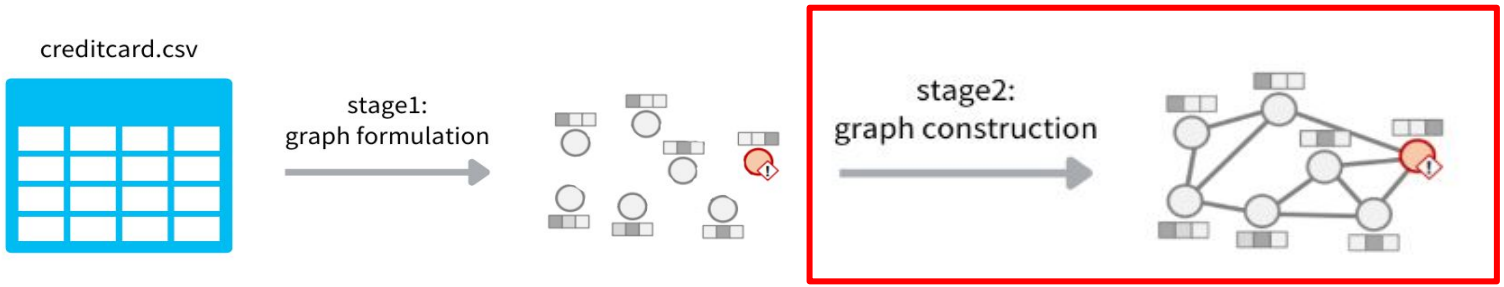


資料表中每一筆交易資料被視為一個節點,

該交易資料的欄位值作為該節點的 features

```
G=nx.Graph()
for i in tqdm(range(df.index),desc="Add nodes into G..."):
    node_name=i
    feature=(df.iloc[i]).to_dict()
    G.add_node(node_name,**feature)
```

2.Graph Construction

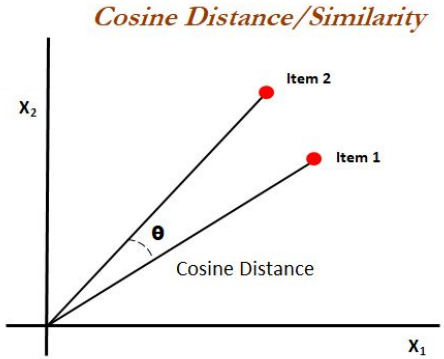


使用**cosine similarity**來當作判斷兩兩節點相似度的標準，並設定一個threshold，高於此門檻則視為兩節點之間有一條邊連通

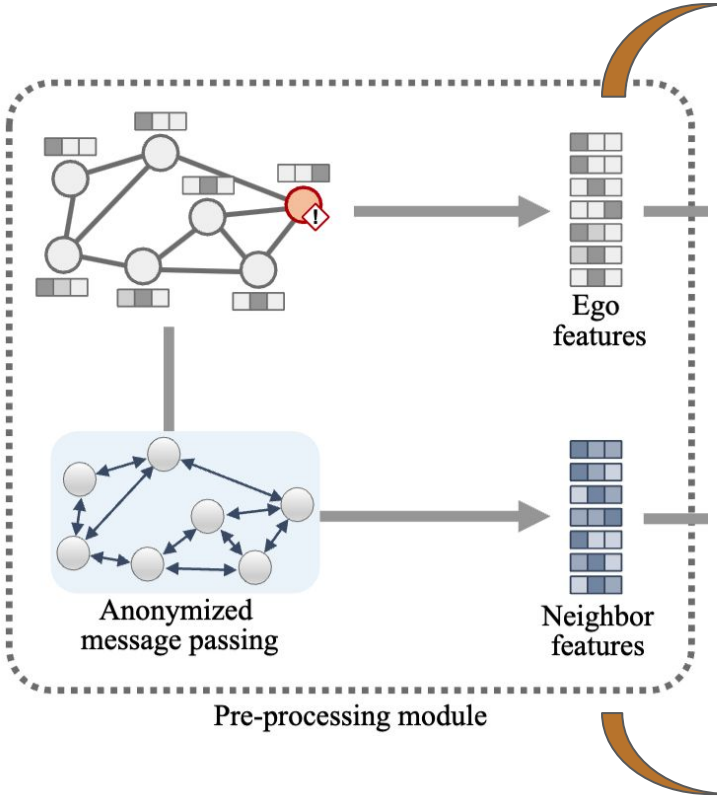
```
for i in tqdm(range(cos_sim.shape[0]),desc="Graph Construction..."):
    for j in range(i+1,cos_sim.shape[1]):
        if cos_sim[i,j]>threshold:
            G.add_edge(i,j)
```

```
for factor in range(31):
    # threshold
    threshold=cos_sim_dict['median']+factor*cos_sim_dict['std']
    data_dic['threshold'].append(threshold)
    G=graph_construction(cos_sim,df,threshold)
    graph.append(G)
    # 存graph
    save_graph(str(threshold), G)
```

Threshold為**cos_similarity的median加上n個標準差**



3.Representation Learning



we construct *ego features* $\mathbf{X}^{(e)}$ by collecting the raw feature vectors together directly, i.e., $\mathbf{X}^{(e)} = \mathbf{X}$. This approach enables us to capture the intrinsic attributes and characteristics of each node without considering the surrounding context.

$$z_i^{(l)} = W^{(l)} h_i^{(l)}, \tag{1}$$

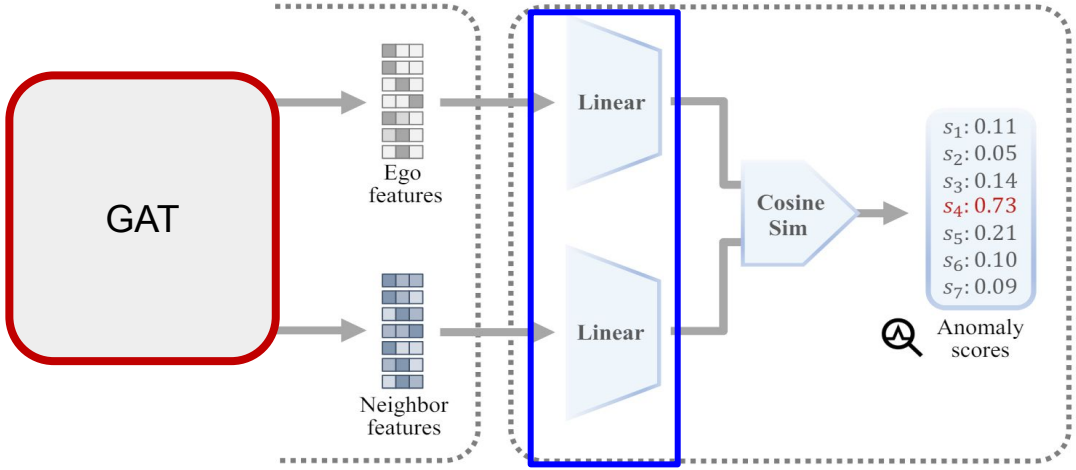
$$e_{ij}^{(l)} = \mathrm{LeakyReLU}(\vec{a}^{(l)^T} (z_i^{(l)} || z_j^{(l)})), \tag{2}$$

$$\alpha_{ij}^{(l)} = \frac{\exp(e_{ij}^{(l)})}{\sum_{k \in \mathcal{N}(i)} \exp(e_{ik}^{(l)})}, \tag{3}$$

$$h_i^{(l+1)} = \sigma\left(\sum_{j \in \mathcal{N}(i)} \alpha_{ij}^{(l)} z_j^{(l)} \right), \tag{4}$$

4.Training Plans

stage4:training plans

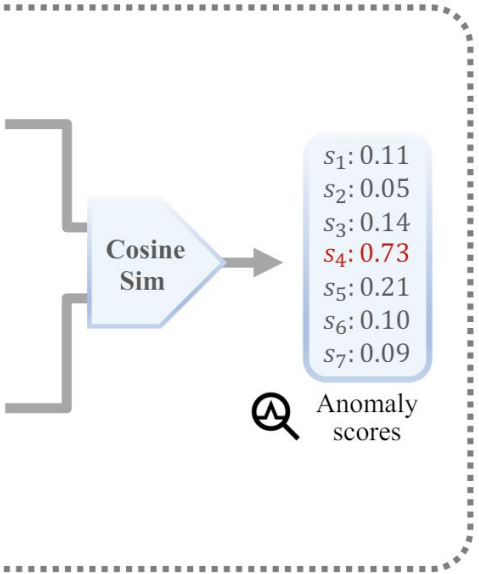


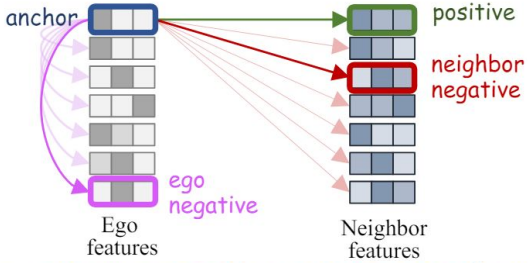

```python
class myGNN(nn.Module):
    def __init__(self, enc_num_heads, enc_input_dim, enc_hidden_dim, enc_num_layers, linear_output_dim):
        super(myGNN, self).__init__()
        # GAT(input、output大小一樣)
        self.encoder_neighbor = Encoder_GAT(enc_num_heads, enc_input_dim, enc_hidden_dim, enc_input_dim, enc_num_layers)
        # linear層
        self.proj_head_neighbor = nn.Linear(enc_input_dim, linear_output_dim)
        self.proj_head_ego = nn.Linear(enc_input_dim, linear_output_dim)

        self.init_emb()
```

4.Training Plans

stage4:training plans





$$c_i^{(pos)} = c_i = \cos(\mathbf{h}_i^{(e)}, \mathbf{h}_i^{(n)})$$

$$c_i^{(neg_{ego})} = \cos(\mathbf{h}_i^{(e)}, \mathbf{h}_k^{(e)})$$

$$c_i^{(neg_{nbr})} = \cos(\mathbf{h}_i^{(e)}, \mathbf{h}_j^{(n)})$$

```
c_neighbor_pos = model.discriminator(h_ego, h_neighbor)
c_neighbor_neg = model.discriminator(h_ego, h_neighbor_neg)
c_ego_neg = model.discriminator(h_ego, h_ego_neg)
```

training loss

$$L = -\sum_{t=1}^{N}(\log(\hat{c}_i^{(pos)}) + \alpha \log(1 - \hat{c}_i^{(neg_{nei})}) + \gamma \log(1 - \hat{c}_i^{(neg_{ego})}))$$

```
loss_sum = loss_pos + args['alpha'] * loss_aug  + args['gamma'] * loss_nod
```

anomaly score

$$s_i = -c_i^{(pos)}$$

```
@staticmethod
def discriminator(x1, x2):
    return -1 * F.cosine_similarity(x1, x2, dim=1).unsqueeze(0)
```

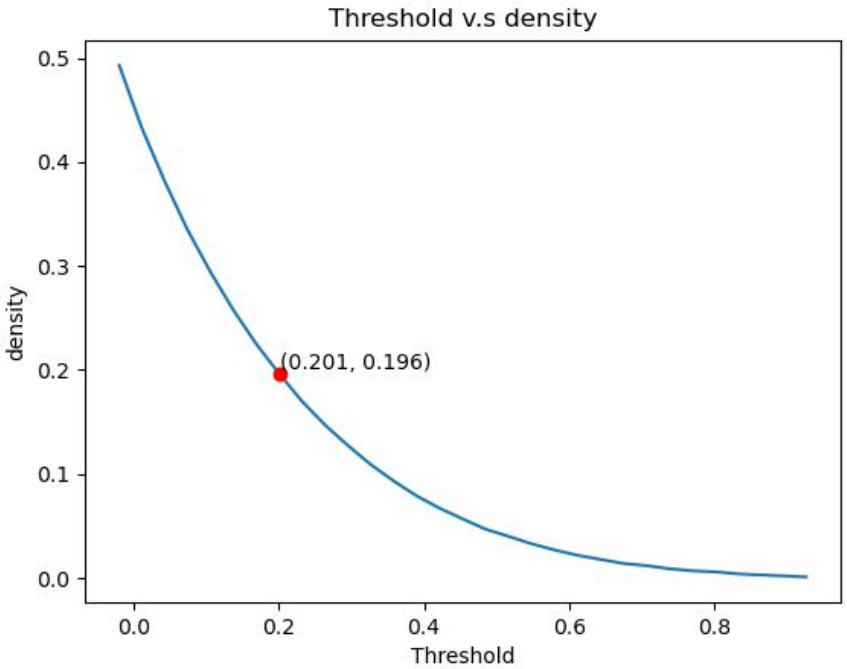
Outline

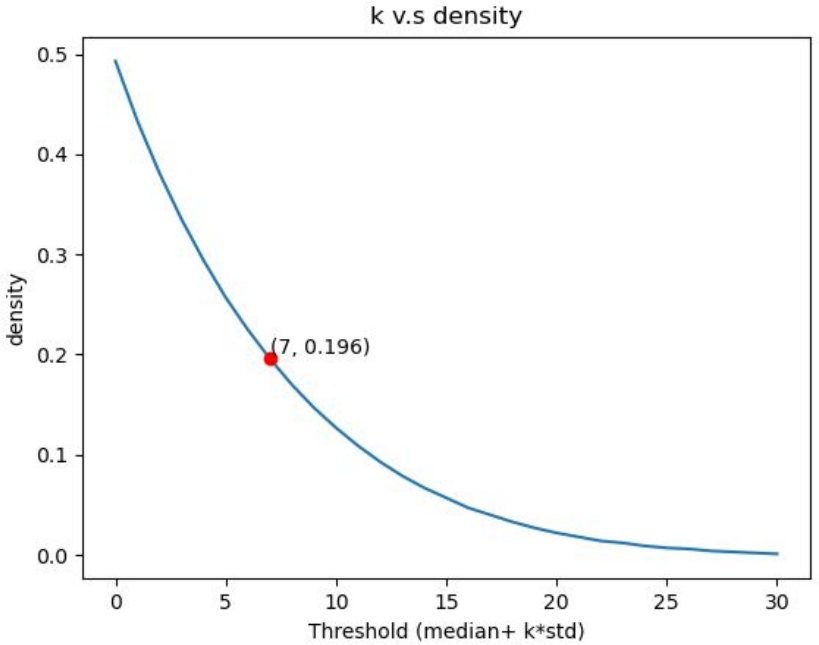
- Problem Definition
- Dataset
- Pipeline
- Method
- **Experiments**
- Future Work & Conclusion

Experiments

1.Threshold v.s Density (undirected graph)

$$Density = \frac{R}{N(N-1)/2}$$

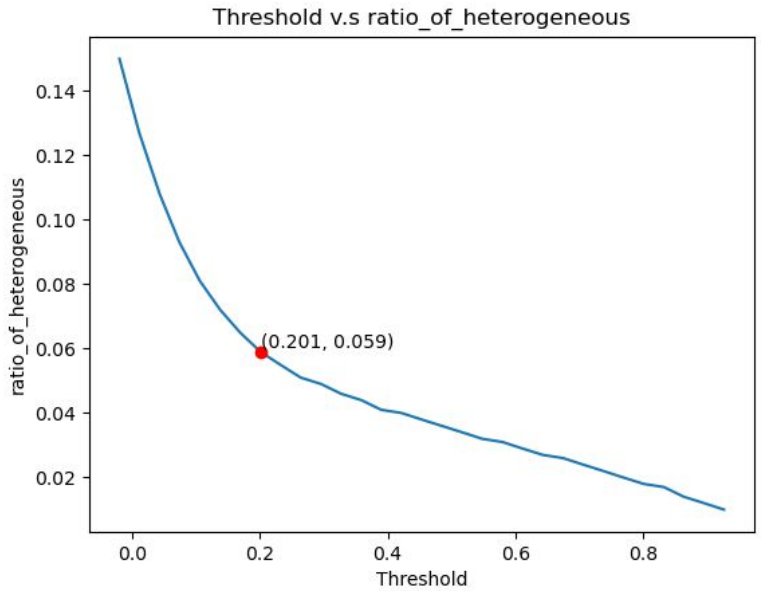


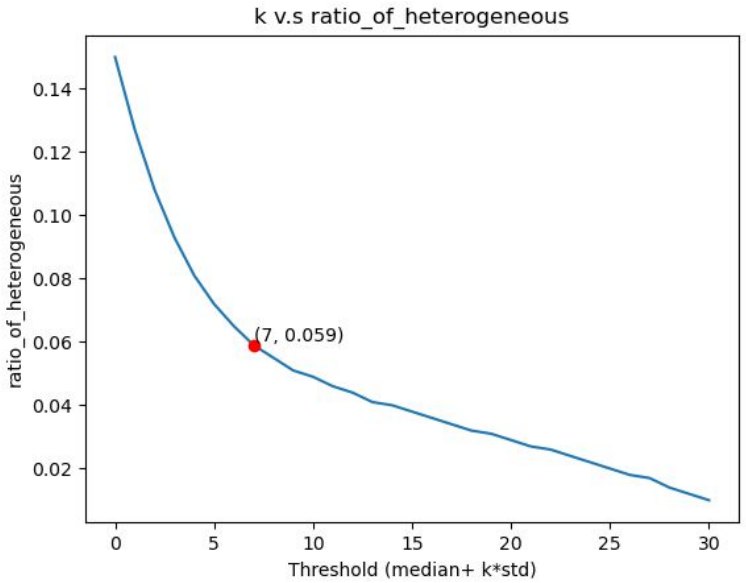


Experiments

2.Threshold v.s Heterogeneous edge ratio (異質邊:連接兩個不同label的節點的邊)

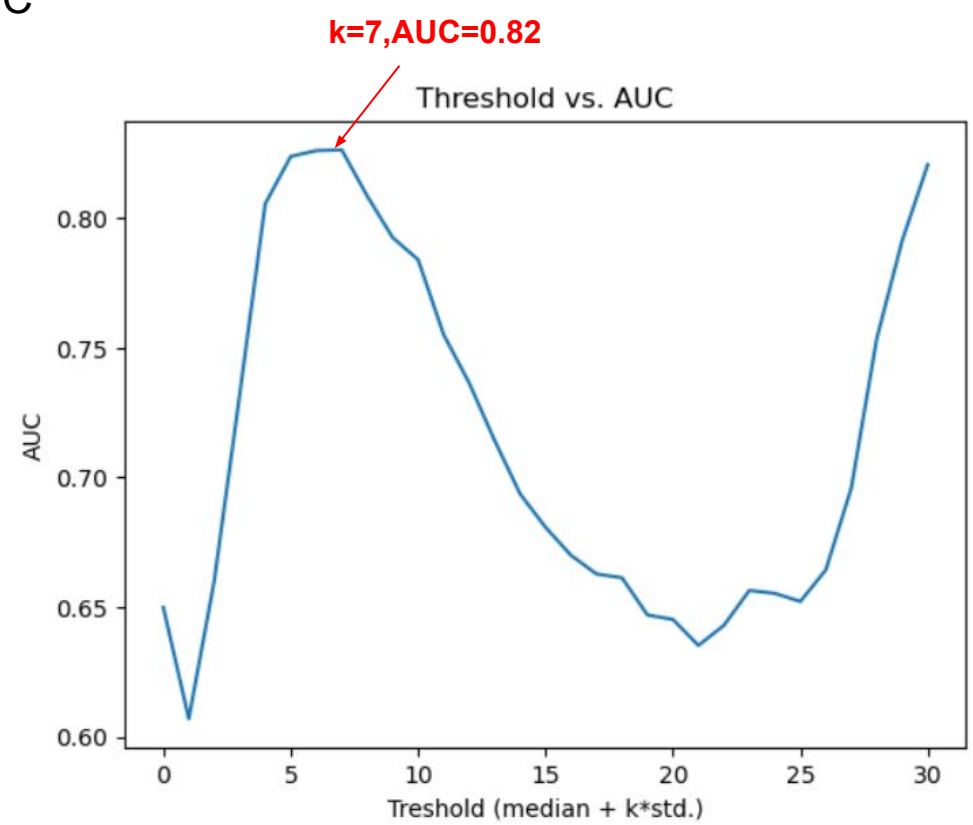
$$\text{Heterogeneous edge ratio} = \frac{NumOfHeterogeneousEdge}{NumOfAllEdge}$$





Experiments

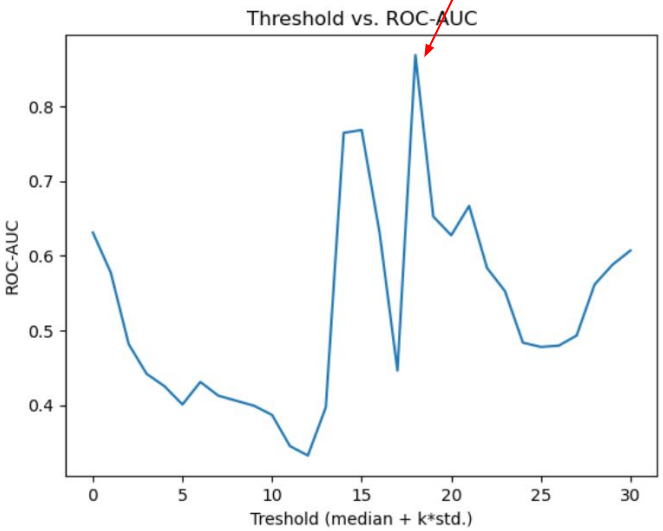
3.Threshold v.s AUC



在訓練集上的結果

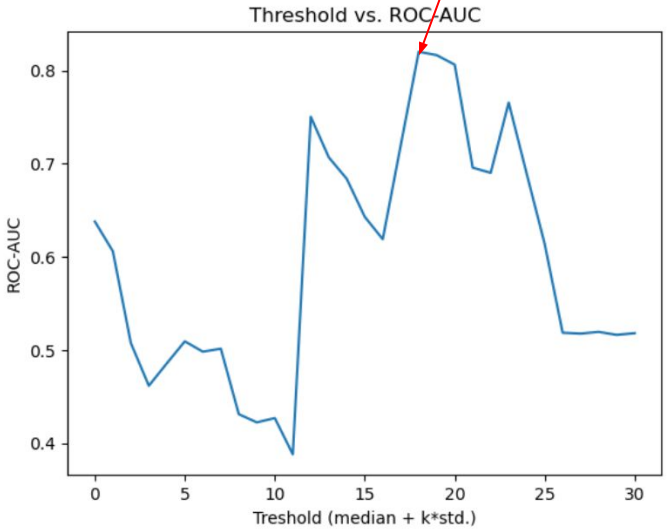
AUC

auc: 0.868



訓練集只有正常資料

auc: 0.820



訓練集混入詐騙資料

Comparison: PyOD異常檢測方法

PyOD(Python Outlier Detection)是一個用於異常檢測的python工具包，裡面有許多經典與新興的異常檢測方法，我們這次採用了以下幾種PyOD提供的方法來與我們的模型進行比較

- KNN(最近的k個鄰居投票)
- LOF(Local Outlier Factor，藉由計算一個樣本點周圍的樣本點所處位置的平均密度與該樣本點本身所在位置密度的比值來推估是否為異常點)
- IForest(孤立森林演算法，建構出的二元樹中，異常樣本會先被孤立出來，離root的平均距離會比正常樣本還要近)
- ECOD(基於經驗累積分布的異常檢測方式，由於異常值是distribution尾端的罕見事件，因此可以透過測量樣本在distributio中的位置來進行異常檢測)
- FeatureBagging(一種ensemble learning的異常檢測方式)
- LUNAR(基於GNN的異常檢測方法)

Comparison: PyOD異常檢測方法

	AUC(fraud資料均不在訓練集)	AUC(fraud資料均勻分布在訓練集與測試集)
KNN	0.9613	0.8856
LOF	0.9611	0.8864
IForest	0.9533	0.9258
ECOD	0.9395	0.9209
FeatureBagging	0.9628	0.8888
LUNAR	0.9601	0.776

Outline

- Problem Definition
- Dataset
- Pipeline
- Method
- Experiments
- **Future Work & Conclusion**

Future Work & Conclusion

1.以基於GNN與對比式學習的baseline:PREM為基礎，加上了GAT，實現了於類別不平衡的表格式資料之非監督式異常檢測，並在 AUC指標上達到接近了 PyOD異常檢測package的表現

2.目前的建邊方式或許較為簡單，未來或許可以在建邊時採用 learning-based的方式，或是在GNN的訊息傳遞階段採用能針對每個節點去個別學習生成的節點圖譜濾波器來進一步強化異常節點資訊的利用 (參考:Partitioning Message Passing for Graph Fraud Detection)

3.在PyOD的比較中，會發現tree類型的算法的確具有很好的表現，我們也可將 tree類算法作為特徵提取工具或是建邊依據的基準，來嘗試將模型表現提升

謝謝大家~