Task

Building a Bitcoin Fraud Classifier using GNN and defend on your results.

Business Statement

The task is to classify Bitcoin transactions as either "licit" or "illicit." The dataset represents a graph where nodes are transactions, and edges indicate flows of Bitcoin. Classifying transactions correctly can help detect fraudulent or suspicious activities.

Actions Taken

1. Dataset Loading: Load the datasets, including node features, edges and classes.
2. Pre-processing: Clean the data, manage missing values, and encode the labels.
3. GNN Model Setup: Implement a Graph Convolutional Network (GCN) using the transaction graph structure.
4. Training: Train the GCN to classify nodes as either licit, illicit, or unknown.
5. Evaluation: Analyse performance metrics like accuracy and loss.

Why GNN?

GNN benefits from capturing the relationships (edges) between transactions, which is crucial for detecting patterns in fraud detection. Traditional ML models like decision trees or SVM ignore the graph structure and treat each node independently, losing the transaction relationships.

Dataset Description

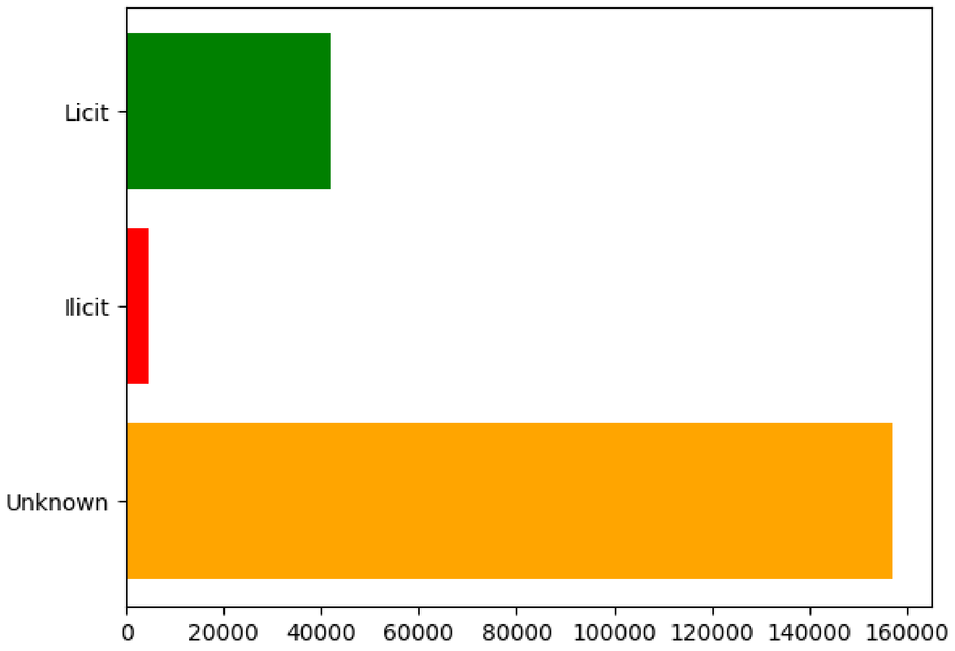
Data collected from here

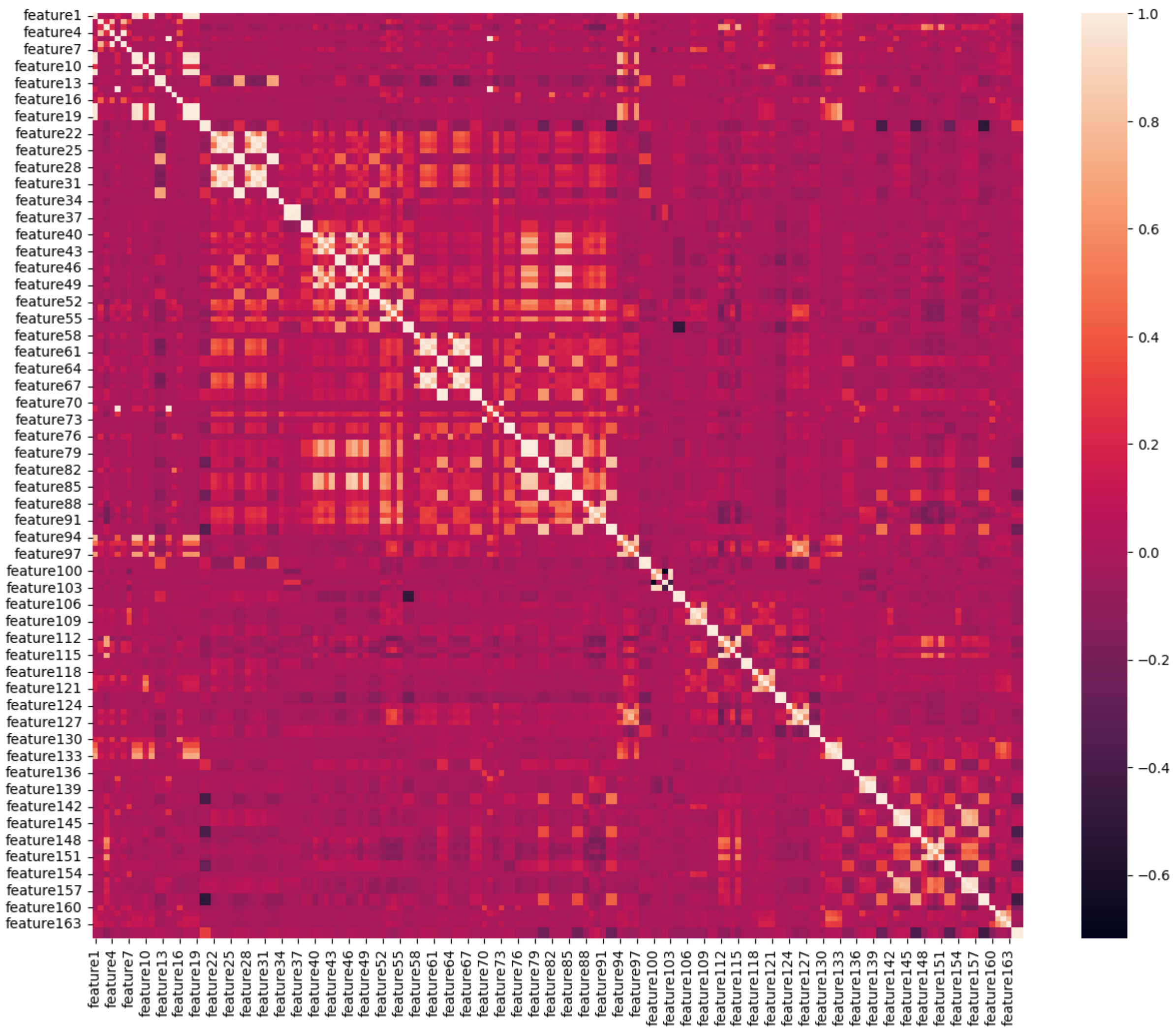
1. Nodes: Transaction ID (203,769 nodes)
2. Edges: Flow of Bitcoin Transactions (234,355 edges)
3. Features: 166 anonymous attributes per node (94 local features, 72 external features)
4. Labels: 1: Illicit, 2: Licit, 0: Unknown

Pre-Processing

The data was already cleaned and properly formatted, additional steps were executed for easier reading, understandability and computation.

1. Replacing unknown labels with "0" (unknown class).
2. Removing transaction records with *NaN*
3. Merging features and classes based on transaction IDs.
4. Converted the data from DataFrame to a Tensor to work with the *torch_geometric* module.


Licit
Illicit
Unknown
0
20000
40000
60000
80000
100000
120000
140000
160000

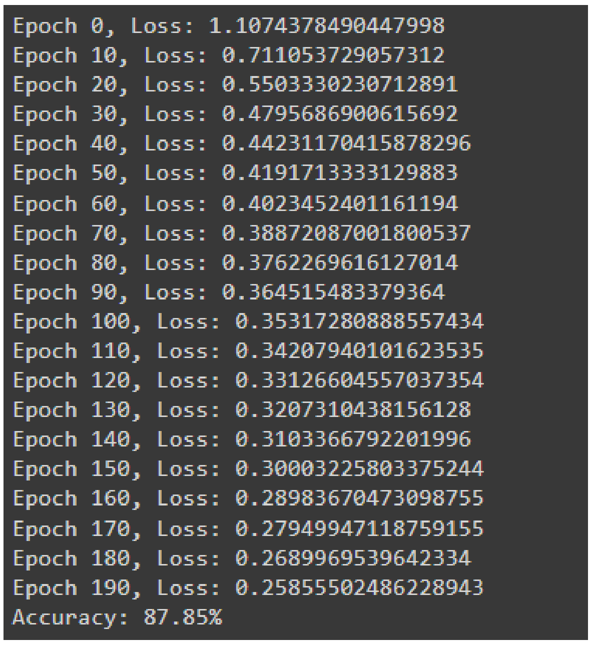

feature1
feature4
feature7
feature10
feature13
feature16
feature19
feature22
feature25
feature28
feature31
feature34
feature37
feature40
feature43
feature46
feature49
feature52
feature55
feature58
feature61
feature64
feature67
feature70
feature73
feature76
feature79
feature82
feature85
feature88
feature91
feature94
feature97
feature100
feature103
feature106
feature109
feature112
feature115
feature118
feature121
feature124
feature127
feature130
feature133
feature136
feature139
feature142
feature145
feature148
feature151
feature154
feature157
feature160
feature163
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6

Training and Analysis

The code performs graph-based classification using a Graph Neural Network (GNN) built on PyTorch Geometric. It starts by converting a networkx graph into a PyTorch Geometric format, followed by assigning node features and labels. The node features are extracted from a DataFrame (df), using columns 2 to 166. Each node is identified by its txId, and node IDs are mapped to their indices within the data. Labels corresponding to the class (0: unknown, 1: illicit, 2: licit) are assigned using a tensor initialized with -1 for unknown nodes. The code checks for missing values (NaNs) in node features, removes rows with NaNs, and updates the edge_index to ensure the edge connections correspond only to valid nodes.

A GNN model is then defined with two GCNConv layers (Graph Convolutional Network layers) that process 166 input features into 64 and 32 dimensions, respectively. The final output is passed through a fully connected layer to classify nodes into 3 classes using log-softmax for prediction. An Adam optimizer and cross-entropy loss are used for training the model over 200 epochs. The training loop updates the model's weights, and the test function evaluates the model by computing prediction accuracy. The final accuracy is printed as a measure of the model's performance in classifying nodes.

The accuracy of 87.85% is a solid performance, reflecting that the model correctly predicts the class labels for a significant portion of the nodes. This implies that the Graph Neural Network is effective in extracting meaningful information from the graph structure and node features



```
Epoch 0, Loss: 1.1074378490447998
Epoch 10, Loss: 0.711053729057312
Epoch 20, Loss: 0.5503330230712891
Epoch 30, Loss: 0.4795686900615692
Epoch 40, Loss: 0.44231170415878296
Epoch 50, Loss: 0.4191713333129883
Epoch 60, Loss: 0.4023452401161194
Epoch 70, Loss: 0.38872087001800537
Epoch 80, Loss: 0.3762269616127014
Epoch 90, Loss: 0.364515483379364
Epoch 100, Loss: 0.35317280888557434
Epoch 110, Loss: 0.34207940101623535
Epoch 120, Loss: 0.33126604557037354
Epoch 130, Loss: 0.3207310438156128
Epoch 140, Loss: 0.3103366792201996
Epoch 150, Loss: 0.30003225803375244
Epoch 160, Loss: 0.28983670473098755
Epoch 170, Loss: 0.27949947118759155
Epoch 180, Loss: 0.2689969539642334
Epoch 190, Loss: 0.25855502486228943
Accuracy: 87.85%
```