Comparative Analysis of Node Classification Models for Fraud Detection on the Ethereum Blockchain Dataset

Anastasiia Golovchenko et al.

Abstract

The rapid expansion of the Ethereum blockchain has presented novel challenges in ensuring the security and authenticity of on-chain transactions. This research explores the possibility of detecting phishing scams on the Ethereum blockchain with graph-based node classification models. The range of models explored are divided into two main groups–full-graph methods and novel ego graph methods. The ego graph methods delve into the local topologies of transactional subgraphs, incorporating attributes such as transaction amount and direction, to cultivate rich node representations. The focus on graph-based methods offers a more granular analysis of transactional relationships, which is critical for identifying subtle patterns indicative of fraud. The utility of the explored approaches are quantified using metrics such as precision and recall in order to analyze proficiency in identifying genuine fraudulent cases amongst an imbalanced dataset. This work underscores the potential of graph-based analytical techniques in preempting and mitigating the risks associated with decentralized digital transactions.

Keywords: Ethereum, Blockchain, Phishing scams detection, Network representation learning, Graph embedding, Ego-graphs.

1. Introduction

Blockchain enables efficient, verifiable, and permanent recording of transactions between parties (Iansiti and Lakhani, 2017) and is expected to bring significant changes across various domains (Yuan and Wang, 2018). Ethereum, one of the most widely adopted blockchain platforms, has seen rapid growth in recent years (Wood et al., 2014). However, the increasing popularity of Ethereum has also attracted attackers, leading to a rise in cybercriminal activities, particularly phishing scams (Holub and O'Connor, 2018). Phishing involves impersonating legitimate websites to deceive users into disclosing sensitive information (Abdelhamid et al., 2014). Traditional phishing detection methods based on identifying suspicious websites cannot be applied due to phishing scams' diverse tactics, such as using high-reward propaganda to directly entice users to send funds (Chainalysis, 2019). These scams have caused substantial financial losses and pose a threat to the security of the Ethereum ecosystem (Conti et al., 2018). The 2024 Crypto Crime Report by Chainalysis reveals that targeted approval phishing scams have grown explosively over the last two years (Chainalysis, 2024), highlighting the need to develop effective methods for identifying and mitigating scams on Ethereum.

This study contributes to the growing body of research by exploring two distinct approaches: one based on analyzing ego-graphs of individual accounts and another that considers the entire transaction graph. By combining advanced graph embedding techniques with machine learning classifiers, this study aims to enhance the performance of phishing detection techniques. The experimental results demonstrate the effectiveness of both approaches.

2. Background and related work

The transparency of public blockchain systems allows for open access to all transaction records (Zheng et al., 2020), which can be represented as a directed transaction network. In such a multidigraph, each node corresponds to a unique address and edges indicate the presence of transfers between two addresses.

Traditional approaches to phishing detection on Ethereum combine machine learning algorithms with manually engineered features, such as structural and statistical properties of nodes. For example, Chen et al. (Chen et al., 2020a) use a LightGBM-based ensemble machine learning algorithm to identify phishing nodes. Alternatively network representation learning techniques can be applied to automatically learn deep features. The most common techniques can be grouped into several categories, including factorization, random walk methods, deep learning, and graph kernels. For instance, Shervashidze et al. (Shervashidze et al., 2011) develop a collection of kernels for large graphs with discrete node labels. Wu et al. (Wu et al., 2020) introduce Trans2Vec, the first network embedding algorithm based on Node2Vec.

Recently, researchers have focused on more advanced graph neural network architectures. Chen et al.(Chen et al., 2020b) present E-GCN, the first application of Graph Convolutional Networks (GCN). Li, S. et al. (Li et al., 2022b) propose a Temporal Transaction Aggregation Graph Network (TTAGN). To address the limitations of using full-size graphs, which can lead to significant computational resource requirements and necessitate handcrafted node features, some studies propose subgraph-based approaches. Adhikari et al. (Adhikari et al., 2018) introduce SubVec, an unsupervised algorithm for learning scalable subgraph feature representations. Lv et al. (Lv et al., 2021) devise an embedding imgraph2vec using subgraphs and a modified Graph2Vec model. Wang et al. (Wang et al., 2021) design a Transaction SubGraph Network (TSGN) classification model that extracts transaction subgraphs around each node and learns subgraph embeddings. Li, P. et al. (Li et al., 2022a) develop a Phishing Detection Graph Neural Network framework (PDGNN) that involves extracting transaction subgraphs of collected phishing accounts.

The existing literature has demonstrated the effectiveness of representing the Ethereum transaction network as a directed graph and that using structural and temporal information from transaction records is effective for phishing detection. However, there is still potential for capturing more complex patterns and developing more scalable and efficient detection models.

3. Methodology

3.1 Data Collection and Preprocessing

This study employs the dataset from the research of Lv and Ding (Lv et al., 2021). The authors obtained over 1600 verified phishing addresses from Etherscan reports, randomly selected an equal number of normal addresses, and collected transaction records for each target address using the API provided by Etherscan. The data included block height, timestamp, transaction amount, transaction information between the target address and its first-level and next-level transaction neighbors, as well as a list of labeled phishing addresses. The collected CSV files are combined into a single DataFrame for further processing and analysis. Once combined, the DataFrame is used to construct a graph of the transaction network where nodes (over 30,000) represent addresses and edges (over 80,000) represent transactions. Figure 1 shows a comparison of transaction behaviors between normal and phishing

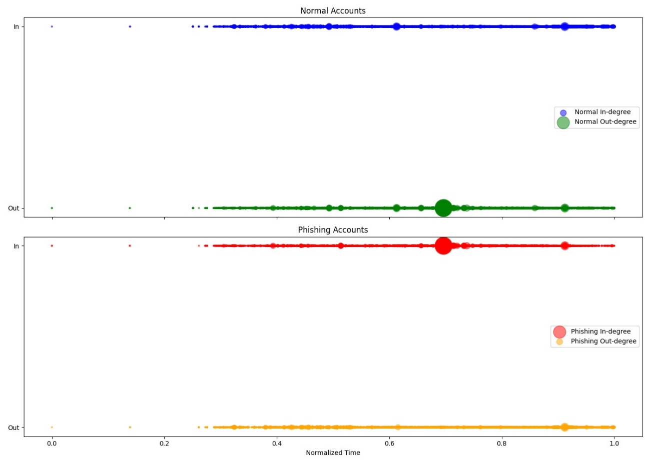

Figure 1: Transaction Behavior Comparison between Normal and Phishing Accounts

accounts over normalized time. Each bubble represents the degree of transaction activity, with bubble size indicating the total transaction value and color representing the direction of the transaction (blue and green for normal accounts, red and orange for phishing accounts). In-degree transactions are depicted above the x-axis, while out-degree transactions are below the x-axis.

4. Methods

4.1 Node2Vec

Node Embedding

The transition probability from node u to node x, $\pi_{ux}(\alpha)$, combines attribute-based and structure-based probabilities (9.5). Post random walk generation, embeddings are learned via the Skip-gram model, which optimizes:

$$\max f \sum_{u \in V} \log Pr(NS(u)|f(u))$$

where f maps nodes to embeddings and $NS(u)$ represents the neighborhood defined by the walks.

Parameter Tuning and Classification

Key parameters such as the dimensions (64), walk length (30), number of walks (200), window size (10), minimum count (1), and batch words (4) are finely tuned to effectively balance the exploration depth and computational efficiency. Using these embeddings, classification tasks are conducted with various machine learning classifiers like logistic regression, SVM, random forest, and KNN, evaluated on precision, recall, F1-score, and accuracy.

4.2 Graph2vec

Graph2vec is a technique that extends the principles of node embedding to entire graphs, facilitating tasks like graph classification and similarity assessment. It captures the essence of graph structures by mapping them into a vector space, where each graph is represented as a fixed-length vector. Overall, Graph2vec offers a comprehensive approach to learning graph representations, enabling effective analysis and classification of graph data in various domains.

Graph Embedding and Learning Objective

Utilizing an embedding function $\phi(G)$, Graph2vec transforms graphs G with nodes V and edges E into continuous vector representations. This process is driven by a neural network f parameterized by Θ, optimizing the graph's structural and topological properties into a dense vector form.

Graph Similarity and Training Objective

The training of Graph2vec is aimed at minimizing the loss function L, which evaluates the difference between the actual structural similarities of graph pairs and their vector representations:

$$L = \sum_{(G_1,G_2) \in D} [sim(G_1, G_2) - sim(\phi(G_1), \phi(G_2))]^2$$

Here, D represents a set of graph pairs used for training, emphasizing the model's ability to discern structural nuances between different graphs.

Integration of Parameters in Model Configuration

To optimize its performance, Graph2vec configures various internal settings, such as embedding dimensions set to 64 to capture sufficient structural detail without overfitting. The model utilizes walk lengths of 10 to adequately explore the graph's architecture and performs 10 walks per node to ensure diverse structural features are captured. The window size for aggregating node context is set to 5, balancing local and broader graph features. A minimum count of 1 ensures that all nodes are considered, and a batch size of 128 optimizes the balance between computational efficiency and memory usage.

4.3 Graph Convolutional Networks (GCNs)

GCNs effectively utilize the graph structure and node features by employing graph convolutional operations to capture neighborhood information. This direct approach bypasses the need for biased random walks, enhancing efficiency for both node and graph-level tasks. Through a stacked architecture of multiple graph convolution layers and the activation of non-linear functions, GCNs propagate node features effectively across the graph. This capability enables them to learn complex patterns and dependencies within the graph structure, making GCNs particularly suitable for nuanced tasks such as phishing detection in network security applications.

Node Embeddings

GCNs extend the concept of convolution to graph data, adapting it for the irregular structure of graphs. By applying convolutional layers to propagate and integrate information across neighborhoods, GCNs extract both local and global contextual information, creating detailed node embeddings. This process is expressed mathematically as:

$$H^{(l+1)} = \sigma\left(D^{-\frac{1}{2}} A D^{-\frac{1}{2}} H^{(l)} W^{(l)}\right)$$

where $H^{(l)}$ denotes the node features at layer l, A is the adjacency matrix, D is the degree matrix, $W^{(l)}$ is the weight matrix at layer l, and σ represents the nonlinear activation function, typically ReLU.

Parameter Considerations

GCNs start with an input feature dimensionality of 64 to balance expressiveness against computational demands. The model's architecture includes graph convolutional layers with hidden dimensions of 32 and 16, designed to progressively capture more abstract representations. A learning rate of 0.01 is chosen to ensure efficient convergence without compromising training stability, and the model trains over 100 epochs, optimizing the learning process without risking overfitting.

4.4 GraphSAGE (Graph Sample and Aggregation)

GraphSAGE is a novel inductive learning method designed to generate node embeddings by sampling and aggregating features from a node's local neighborhood. This approach allows the model to efficiently generalize to unseen nodes, making it ideal for dynamic graphs. In essence, GraphSAGE leverages a systematic approach to learn robust node embeddings, which are critical in transaction networks for identifying fraudulent activities.

Sampling and Aggregation

GraphSAGE starts with a fixed-size neighborhood sampling strategy around each node, ensuring the model consistently incorporates relevant information from adjacent nodes. Using graph convolutional layers, the method aggregates these features to produce embeddings. The aggregation function is defined as:

$$h_v^k = \mathrm{AGG}\left(\{h_u^{k-1}, \forall u \in N(v)\}\right)$$

where h_v^k is the node representation of v at layer k, $N(v)$ is the neighborhood of node v, and h_u^{k-1} is the representation of neighbor node u at the previous layer.

Model Architecture and Parameters

The model features an input layer with a dimensionality of 64 (*input_dim*), allowing it to capture essential node attributes without overfitting. It progresses through layers with dimensions set to 128 and 256 (*hidden_dim*) to abstract higher-level features. The learning process utilizes a learning rate of 0.01 (*lr*) and trains over 10 epochs (*num_epochs*), balancing convergence speed and training stability. The GraphSAGE model uses two SAGEConv layers and trains using the Adam optimizer.

Inductive Learning and Classification

GraphSAGE's inductive learning capability enables it to infer embeddings for new nodes by applying learned patterns from existing nodes. This feature is utilized in node classification tasks using logistic regression, SVM, random forest, and KNN, assessed by precision, recall, F1-score, and accuracy metrics.

4.5 Trans2Vec

The implementation follows the key steps from (Wu et al., 2020): The first step is to perform biased random walks on the transaction network to capture the neighborhood information of nodes. The biased random walk strategy incorporates the transaction amount and timestamp biases. (9.5) The balance parameter α allows for adjusting the bias between the transaction amount and timestamp, providing flexibility in capturing the importance of each factor in the transaction network. After generating the biased random walks, the Word2Vec model from the gensim library is employed to learn the node embeddings. The Word2Vec model is based on the Skip-gram architecture, which aims to maximize the log probability of observing the neighboring nodes given the embedding of the center node 9.1, after which the learned node embeddings are then evaluated for node classification tasks using various classifiers.

4.6 Subgraph Embeddings

Constructing k-hop Subgraphs

Given a directed graph $G = (V, E)$ where V is the set of nodes and E is the set of directed edges between the nodes, we define a k-hop subgraph centered at node v_i as $SG(v_i, k)$. This subgraph includes all nodes v_j such that the shortest path from v_i to v_j is at most k edges, and all edges $(v_m, v_n) \in E$ for which v_m and v_n are within this k-hop neighborhood. Mathematically, the k-hop subgraph $SG(v_i, k)$ is represented as:

$$SG(v_i, k) = (V_i', E_i')$$

$$\text{where: } V_i' = \{v_j \in V | \text{distance}(v_i, v_j) \leq k\}$$
$$E_i' = \{(v_m, v_n) \in E | v_m, v_n \in V_i'\}$$

The adjacency matrix A_{SG} of the subgraph $SG(v_i, k)$ is a submatrix of the adjacency matrix A of the entire graph G, containing only the nodes within V_i' and the corresponding edges in E_i'.

4.6.1 Method I: Graph Convolutional Network (GCN) Approach

Feature Augmentation Feature augmentation boosts the GCN's ability to represent Ethereum transactions by leveraging structural, transactional, and interaction features. Structural Features (SF) encapsulate the transaction connectivity of nodes. Transactional Values (TV) distinguish nodes based on transaction amounts, indicating economic significance. Interaction Intensity (II) measures the strength of transactional activity, both incoming and outgoing, relative to the node's structural properties. Details of these computations are outlined in the Appendix 9.2. These features collectively enhance the model's predictive performance in classifying nodes within the Ethereum network

Ego Graph Centric Feature Integration The feature augmentation process enriches nodes within k-hop subgraphs by assigning initial features based on local connections. Central nodes receive a comprehensive set of features (SF, TV, II), while surrounding nodes only get Structural Features to maintain focus. These features are normalized to create a consistent feature matrix X for GCN analysis, allowing nuanced examination of network behavior and phishing activities. Additionally, a focused subgraph generation strategy is implemented: General overview (Method 1) considers all nodes. Focus on phishing and high-influence nodes using Katz centrality (Eq. 10) (Method 2). Addressing imbalance by combining non-phishing nodes with upsampled phishing nodes (Method 3) to potentially improve performance.

Representation Learning for the Subgraphs The Graph Convolutional Network (GCN) learns from the feature matrix X, composed of Structural Features (SF), Transactional Values (TV), and Interaction Intensity (II) within k-hop subgraphs, guiding a thorough analysis of the Ethereum network. This learning process involves iterative updates to node embeddings, captured by Equations 11, 12, and 13, where $H^{(l)}(v)$ denotes the node embedding at layer l. The readout function Readout(G), described in Equation 12, condenses these embeddings to a graph-level representation, facilitating downstream tasks such as graph classification. Finally, optimization of the GCN model utilizes a cross-entropy loss function, as defined in Equation 13, ensuring convergence of predicted graph labels ($\hat{y}_i$) to true labels (y_i) across the Ethereum network.

4.6.2 Method II: Graph2Vec Approach

Feature Augmentation for Graph2Vec For the Graph2Vec model, we assess the transaction amounts with Value-based Edge Metrics to gauge value based engagement, utilize Activity Ratio Metric to analyze the frequency and volume of transactions, and employ Directional Intensity Label to determine the predominant transaction direction of nodes within their subgraphs. For a complete understanding of the feature construction, refer to equations in the Appendix, Section 9.3 for the Graph2Vec features.

Ego Graph Centric Feature Integration Our graph embedding model creates k-hop ego-graphs for every node, encompassing both phishing and non-phishing nodes. Each node receives labels based on transactional attributes, resulting in 12 features. The central node in an ego-graph is labeled '0', highlighting its significance. We also construct a second set of k-hop subgraphs focusing on phishing nodes and high-interaction non-phishing nodes identified via Katz centrality. Feature augmentation enhances node features in both contexts, offering insights into general and high-interaction node behaviors.

Representation Learning for the Subgraphs After relabeling nodes in input ego-graphs, we proceed to representation learning using the Graph2Vec method. Rooted subgraphs meeting our criteria form a corpus for Weisfeiler-Lehman graph relabeling, enriching contextual information. Subsequently, each relabeled subgraph is transformed into a comprehensive graph embedding using a skip-gram model, preserving topological features in the embedding space. These embeddings encapsulate structural and transactional features, capturing the essence of local neighborhoods around central nodes.

5. Experimental design

All methods described above are evaluated by conducting 10 experiments each. For each experiment, a random train/test split is conducted. The split for the full-graph methods are performed on the nodes. The splits for the ego graph methods are performed on the subgraphs. Since there is a large class imbalance, Synthetic Minority Oversampling Technique (SMOTE) is employed for the full-graph methods to address this issue. For the ego graphs Graph2Vec method, upsampling of the subgraphs with phishing nodes as central nodes is carried out. No corrections of class imbalance are included for the ego graphs with GCN. After appropriate feature scaling is performed, all embedding methods are then used with four separate classifiers–Logistic Regression, k-Nearest-Neighbor (kNN), support vector machine (SVM), and random forest– on a binary classification task. Metrics such as accuracy, precision, recall, and f1-score are recorded for each method, classifier, and binary-class combination. For the ego graph GCN method, the method, classifier, and binary-class combination is extended to include which set of subgraphs are included.

6. Experimental Results

The GCN model demonstrates the strongest performance in classifying phishing and non-phishing nodes. For Class 0 (non-phishing), the GCN achieves outstanding metrics with an accuracy of approximately 98.87%, precision of 99.53%, recall of 99.26%, and an F1-Score of 99.39% (Table 5). In Class 1 (phishing), the GCN model maintains high accuracy at 98.87%, with precision, recall, and F1-Score values around 71.43%, 79.37%, and 74.91% respectively. The discrepancy in scores between phishing and non-phishing nodes highlights the negative impact class imbalance has on model performance. All classifiers using full-graph embedding methods perform extremely poorly at both capturing all phishing nodes (low recall) and correctly classifying them as phishing nodes (low precision). GraphSAGE shows similar deficiencies. For ego graph embeddings, classifiers employing Method II embeddings drastically outperforms Method I's combination of embeddings and GCN. While incorporating Katz centrality into Method II slightly improves recall and precision for phishing nodes, it has no marked impact on Method I. Furthermore, upsampling phishing subgraphs for Method II hurts overall performance for a majority of the classifiers used. Individual bar charts in Figures 8, 7, 4, 5, 6, and 3) show model performance across various breakdowns of classifiers, class, and ego graph subsets.

Classifier	Class	Accuracy	Precision	Recall	F1-Score
KNN	0	0.958721	0.991371	0.966144	0.978592
KNN	1	0.958721	0.315448	0.649264	0.424050
Logistic Regression	0	0.877384	0.991671	0.881856	0.933538
Logistic Regression	1	0.877384	0.123067	0.690942	0.208831
Random Forest	0	0.967864	0.990749	0.976210	0.983425
Random Forest	1	0.967864	0.384845	0.619502	0.474362
SVM	0	0.946116	0.993339	0.951204	0.971811
SVM	1	0.946116	0.265573	0.733937	0.389723

Table 1: Classifier Performance Metrics (Trans2vec)

Classifier	Class	Accuracy	Precision	Recall	F1-Score
KNN	0	0.953986	0.979008	0.973675	0.976329
KNN	1	0.953986	0.164673	0.198294	0.178809
Logistic Regression	0	0.828249	0.993721	0.829004	0.903845
Logistic Regression	1	0.828249	0.108964	0.799296	0.191608
Random Forest	0	0.955084	0.984426	0.969261	0.976777
Random Forest	1	0.955084	0.258733	0.411514	0.315963
SVM	0	0.948962	0.979498	0.967902	0.973660
SVM	1	0.948962	0.152485	0.222817	0.180110

Table 2: Classifier Performance Metrics (Node2vec)

Classifier	Class	Accuracy	Precision	Recall	F1-Score
GraphSAGE	0	0.98022	0.982012	0.998049	0.989965
GraphSAGE	1	0.98022	0.704713	0.201721	0.312956

Table 3: Classifier Performance Metrics (GraphSage)

Classifier	Class	Accuracy	Precision	Recall	F1-Score
K-Nearest Neighbors	0	0.919668	0.994431	0.922985	0.957375
K-Nearest Neighbors	1	0.919668	0.188539	0.775208	0.303205
Logistic Regression	0	0.867039	0.981265	0.880791	0.928314
Logistic Regression	1	0.867039	0.049803	0.270195	0.084069
Random Forest	0	0.953308	0.990790	0.961182	0.975737
Random Forest	1	0.953308	0.270850	0.611462	0.372267
Support Vector Machine	0	0.974042	0.981704	0.991930	0.986790
Support Vector Machine	1	0.974042	0.361840	0.198013	0.255360

Table 4: Classifier Performance Metrics (Graph2vec)

Classifier	Class	Iteration	Accuracy	Precision	Recall	F1-Score
GCN	0	4.5	0.988177	0.995256	0.992642	0.993945
GCN	1	4.5	0.988177	0.714268	0.793681	0.749082

Table 5: Classifier Performance Metrics (GCN)

Classifier	Class	Egograph Type	Accuracy	Precision	Recall	F1-Score
ego_gcn	0	full	0.97480	0.975626	0.999117	0.987231
ego_gcn	0	katz	0.82145	0.824797	0.991312	0.900299
ego_gcn	1	full	0.97480	0.387013	0.023710	0.042421
ego_gcn	1	katz	0.82145	0.492991	0.068805	0.118070

Table 6: Classifier Performance Metrics (Egograph GCN)

Classifier	Class	Egograph Type	Accuracy	Precision	Recall	F1Score
KNN	0	full no upsampling	0.978441	0.983804	0.994312	0.989029
KNN	0	full_upsampling	0.847302	0.992136	0.850507	0.915871
KNN	0	katz	0.724719	0.724653	0.714168	0.719100
KNN	1	full_no_upsampling	0.978441	0.545742	0.294917	0.381825
KNN	1	full_upsampling	0.847302	0.099155	0.709646	0.173918
KNN	1	katz	0.724719	0.724890	0.735369	0.729832
Logistic Regression	0	full_no_upsampling	0.978209	0.978817	0.999332	0.988967
Logistic Regression	0	full_upsampling	0.707235	0.993012	0.705416	0.824854
Logistic Regression	0	katz	0.606742	0.593493	0.649436	0.619774
Logistic Regression	1	full_no_upsampling	0.978209	0.703024	0.067557	0.122948
Logistic Regression	1	full_upsampling	0.707235	0.058237	0.786052	0.108418
Logistic Regression	1	katz	0.606742	0.623017	0.565834	0.592590
Random Forest	0	full_no_upsampling	0.979112	0.980382	0.998611	0.989412
Random Forest	0	full_upsampling	0.974807	0.985916	0.988342	0.987127
Random Forest	0	katz	0.758427	0.756593	0.754266	0.754870
Random Forest	1	full_no_upsampling	0.979112	0.704708	0.138721	0.230508
Random Forest	1	full_upsampling	0.974807	0.437678	0.391728	0.412747
Random Forest	1	katz	0.758427	0.761069	0.763432	0.761707
SVM	0	full_no_upsampling	0.978837	0.979515	0.999244	0.989281
SVM	0	full_upsampling	0.810595	0.993640	0.811402	0.893305
SVM	0	katz	0.734270	0.699137	0.810620	0.750547
SVM	1	full_no_upsampling	0.978837	0.755370	0.098791	0.174390
SVM	1	full_upsampling	0.810595	0.087150	0.776306	0.156644
SVM	1	katz	0.734270	0.781343	0.660129	0.715424

Table 7: Classifier Performance Metrics (Egograph Graph2vec)

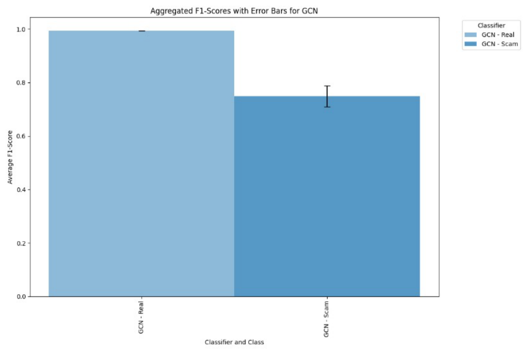

Figure 2: GCN

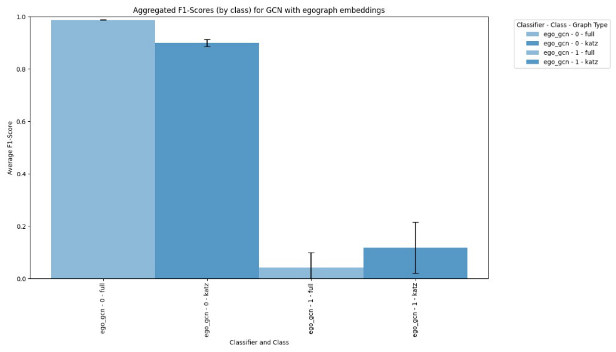

Figure 3: Ego Graph GCN

7. Conclusion

This study presents a comprehensive analysis of various graph-based machine learning techniques applied to the classification of phishing and non-phishing nodes within Ethereum transaction data. The experimental results demonstrate that the Graph Convolutional Network (GCN) model outperforms other methods in distinguishing between the two classes. The use of GCN models showcase their potential in capturing the intricate patterns and relationships inherent in transactional data. Notably, the Ego Graph G2V approach also yields promising results. It highlights the value of capturing local neighborhood information around central nodes to enhance classification accuracy. Despite not surpassing the GCN model, it offers a valuable perspective on the utility of localized features in graph-structured data.

8. Limitation and Future Work

The issue of class imbalance in Ethereum transaction block data significantly challenges the predictive modeling process. To address this, the development of novel synthetic data generation techniques, such as GraphSMOTE, is essential. GraphSMOTE, an extension of SMOTE to graph-structured data, can intelligently oversample the minority class (e.g., phishing nodes) by creating synthetic but realistic nodes within the graph structure. Furthermore, the concept of buffer nodes presents an intriguing research direction. Buffer nodes, or intermediary nodes inserted between two genuine nodes, can regulate the flow of information within the network. These nodes can act as gatekeepers, controlling the transfer of data and potentially mitigating the rapid spread of fraudulent activity. By monitoring transactions through these buffer nodes, it's possible to isolate suspicious patterns before they propagate through the network. Investigating graph neural network architectures that can incorporate buffer nodes within their layers could lead to innovative ways to model and predict transactions, adding an additional layer of security to detect anomalous behavior. Incorporating buffer nodes could also enhance link prediction models. By examining the flow of transactions through buffer nodes, these models can learn the typical transaction patterns and better predict the formation of new, legitimate links while flagging anomalies that deviate from these patterns. Moreover, the integration of unsupervised and semi-supervised learning models can further exploit the vast amounts of unlabeled data in blockchain networks. These models can discern the underlying structure and transaction dynamics, providing a stronger base for the supervised detection of phishing activities. These strategies signify a holistic approach to enhancing the detection systems for blockchain transactions.

9. Appendix

9.1 Trans2vec

$$PT_{ux} = \frac{T(u,x)}{\sum_{x' \in V_u} T(u,x')} \tag{1}$$

where f is the mapping function from nodes to their embeddings, and $N_S(u)$ represents the neighborhood of node u obtained through the biased random walk strategy S.

9.2 Features for Ego Graphs GCN

1. **Structural Features (SF)**: To capture the local structural properties of nodes, we define two measures of connectivity:

$$SF_{in}(v) = \sum_{u \in V} |E_{vu}|, \quad SF_{out}(v) = \sum_{u \in V} |E_{uv}| \tag{2}$$

   where SF_{in} represents the number of times a node v is at the receiving end of the transaction, and SF_{out} the number of times transactions are outgoing from a node v.

2. **Transactional Values (TV)**: We characterize the financial dynamics between nodes by observing the Ethereum transactions, using the transaction amounts to differentiate the nodes:

$$TV_{ratio}(v) = \frac{\max_{u \in V}(E_{vu})}{\sum_{u \in V} |E_{vu}|}, \quad TV_{out-ratio}(v) = \frac{\max_{u \in V}(E_{uv})}{\sum_{u \in V} |E_{uv}|} \tag{3}$$

   where TV_{ratio} captures the ratio of the highest single transaction amount to the total incoming transactions for node v, and similarly, $TV_{out-ratio}$ for outgoing transactions.

3. **Interaction Intensity (II)**: Reflecting the interaction strength, we compute the normalized interaction intensity, accounting for the transaction count and volume:

$$II_{in}(v) = \frac{\sum_{u \in V, E_{vu} \in E} |E_{vu}|}{SF_{in}(v)}, \quad II_{out}(v) = \frac{\sum_{u \in V, E_{uv} \in E} |E_{uv}|}{SF_{out}(v)} \tag{4}$$

   with II_{in} and II_{out} denoting the intensity of incoming and outgoing interactions respectively, normalized by the structural dynamics measures.These augmented features capture the multi-faceted nature of nodes within the Ethereum network, providing a rich set of descriptors for the GCN to learn from.

9.3 Features for Ego Graphs Graph2Vec

1. **Value-based Edge Metrics (VEM)**: We evaluate the transaction amounts associated with each node to reflect its economic engagement within the ego subgraph:

$$VEM_{avg}(c) = \frac{\sum_{v \in V, v \neq c} wa(v)}{|V| - 1}, \tag{5}$$

   where $wa(v)$ denotes the amount-based weight for each node v, and c is the central node of the subgraph. This measure calculates the average transaction amount that is either sent or received by the neighbors within the subgraph, excluding the central node.

   To classify the edges in the subgraph according to their transaction value, we apply the following rule:

$$ya_{VEM}(v) = \begin{cases} 1, & \text{if } wa(v) \geq VEM_{avg}(c) \\ 0, & \text{otherwise,} \end{cases} \tag{6}$$

   where $ya_{VEM}(v) = 1$ indicates that the transaction amount associated with node v is higher or equal to the average transaction value in the subgraph, signifying a higher contribution to the subgraph's transaction value, and $ya_{VEM}(v) = 0$ indicates a lower contribution compared to the average.

2. **Activity Ratio Metric (ARM)**: This metric measures the frequency and amount of transactions, providing insight into the transactional behavior of nodes:

$$dn_{avg}(c) = \frac{\sum_{v \in V_i', v \neq c} wn(v)}{|V_i'| - 1}, \tag{7}$$

where $wn(v)$ is the number-based weight reflecting the number of transactions involving node v, and c is the center node of the ego-graph $SG(v_i, k)$. This metric captures the average transaction count in the subgraph surrounding node c, providing a baseline to compare each node's activity against.

With this average, we categorize the transaction activity of each node in the subgraph:

$$yn_{ARM}(v) = \begin{cases} 1, & \text{if } wn(v) \geq dn_{avg}(c) \\ 0, & \text{otherwise,} \end{cases} \tag{8}$$

where $yn_{ARM}(v) = 1$ indicates that node v has a transaction count equal to or higher than the average, suggesting active transaction engagement within the ego-network. Conversely, $yn_{ARM}(v) = 0$ signifies a transaction count below the subgraph average, reflecting lesser engagement.

3. **Directional Intensity Label (DIL)**: The Directional Intensity Label (DIL) encodes the net directionality of a node's transactions within its subgraph, indicating whether the node primarily sends or receives transactions:

$$DIL(v) = \text{sign}\left(\sum_{u \in \mathcal{N}_v} E_{vu} - \sum_{u \in \mathcal{N}_v} E_{uv} \right), \tag{9}$$

where sign(x) returns 1 if $x > 0$, indicating that node v is a net sender, -1 if $x < 0$, indicating a net receiver, and 0 if $x = 0$, denoting balanced transactions in and out of node v. Here, $\mathcal{N}_v$ is the set of nodes in the ego-network of v, and E_{vu} and E_{uv} are the transaction values from u to v and from v to u, respectively.

9.4 Katz Centrality Equation

The Katz centrality for a node i in a graph is calculated as:

$$C_K(i) = \sum_{j=1}^{n} \sum_{k=1}^{\infty} \beta^k (A^k)_{ji} \tag{10}$$

where:

- $C_K(i)$ is the Katz centrality of node i,
- β is a parameter that controls the influence of indirect connections,
- A is the adjacency matrix of the graph, and
- n is the number of nodes in the graph.

9.5 Biased Random Walk

The transition probability from node u to node x, $\pi_{ux}(\alpha)$, combines attribute-based and structure-based probabilities:

$$\pi_{ux}(\alpha) = P_A(u,x)^{\alpha} \cdot P_S(u,x)^{1-\alpha}$$

Here, $P_A(u,x)$ focuses on transaction amounts and $P_S(u,x)$ on transaction times, calculated as:

$$P_A(u,x) = \frac{A(u,x)}{\sum_{x' \in V_u} A(u,x')}, \quad P_T(u,x) = \frac{T(u,x)}{\sum_{x' \in V_u} T(u,x')}$$

9.6 Equations for Representation Learning

The learning process in the GCN involves updating the node embeddings through a series of transformations. At each layer l, the node embedding $H^{(l)}(v)$ is computed as:

$$H^{(l)}(v) = \sigma\left(W^l \cdot \text{CONCAT}\left(H^{(l-1)}(v), \bigoplus_{u \in \mathcal{N}(v)} H^{(l-1)}(u)\right) + b^l\right) \tag{11}$$

Here, σ denotes a non-linear activation function, W^l is the weight matrix, b^l is the bias term, and $\mathcal{N}(v)$ represents the neighbors of node v. The operator $\bigoplus$ indicates an aggregation function, such as sum, mean, or max, which combines the features of neighboring nodes.

After the embeddings have been learned, a readout function R is used to summarize the embeddings for the entire graph. This readout function, denoted as Readout(G), is expressed as:

$$\text{Readout}(G) = R\left(\{H^{(L)}(v) \mid v \in V\}\right) \tag{12}$$

Here, $H^{(L)}(v)$ represents the final embedding of node v after L layers, and V denotes the set of all nodes in the subgraph. The readout function can take various forms, such as sum, mean, or more complex graph-level pooling operations, to produce a graph-level representation suitable for downstream tasks like graph classification.

Finally, the optimization of the GCN model involves utilizing a cross-entropy loss function for classification:

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)\right] \tag{13}$$

Here, N is the number of graphs, y_i represents the true label, and $\hat{y}_i$ is the predicted label by the GCN for the i^{th} graph. This objective ensures that the model's predictions converge to the true graph labels, enabling effective learning of graph representations tailored to the specificities of the Ethereum network.

9.7 Experimental Results

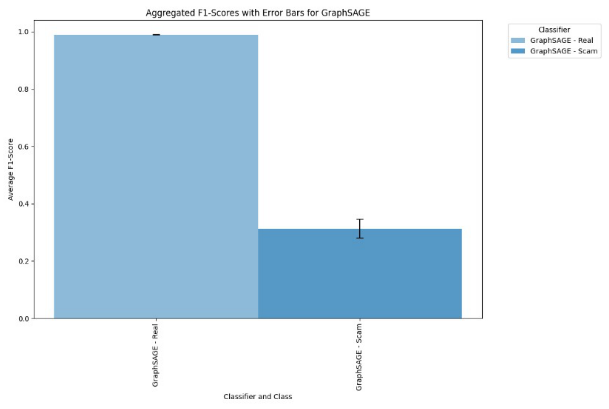

Figure 4: GraphSage

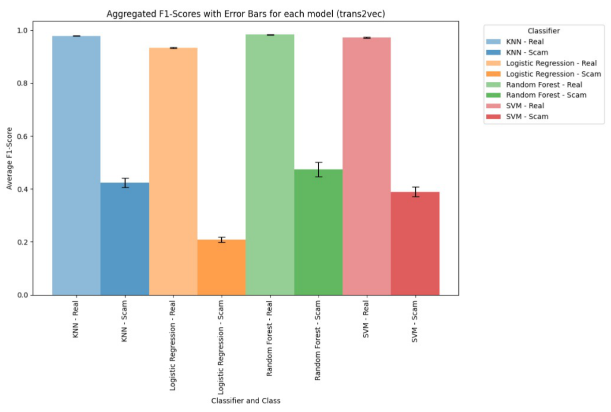

Figure 5: Trans2vec

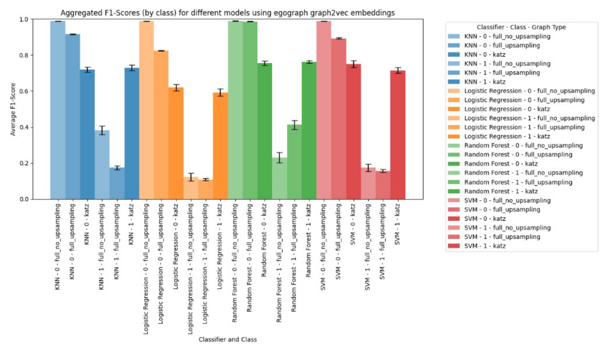

Figure 6: EgoGraph Graph2vec

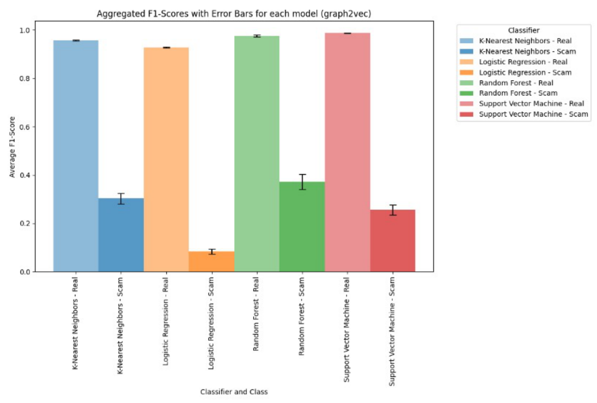

Figure 7: graph2vec

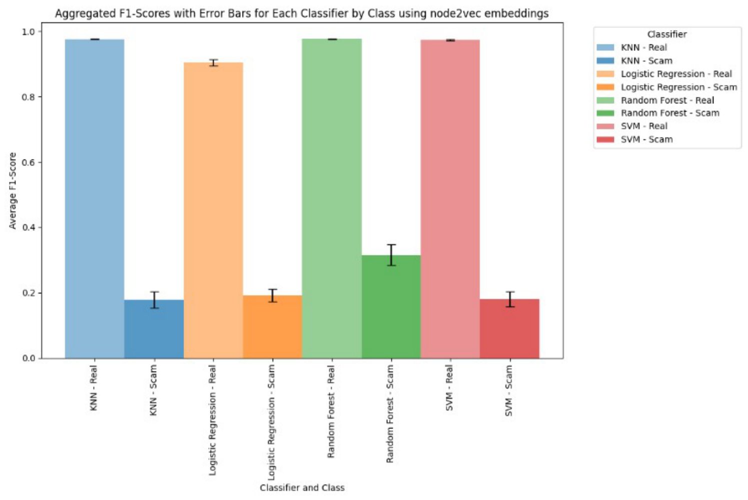

Figure 8: Node2vec

References

Neda Abdelhamid, Aladdin Ayesh, and Fadi Thabtah. Phishing detection: A recent intelligent machine learning comparison based on models content and features. *2014 IEEE International Conference on Intelligent Systems and Control (ISCO)*, pages 1–6, 2014.

Bijaya Adhikari, Yingyu Zhang, Naren Ramakrishnan, and B Aditya Prakash. Sub2vec: Feature learning for subgraphs. *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 170–182, 2018.

Chainalysis. Ethereum scams: How to identify and protect against them, 2019. (`https://blog.chainalysis.com/reports/ethereum-scams`).

Chainalysis. Crypto crime report 2024, 2024. (`https://go.chainalysis.com/rs/503-FAP-074/images/The%202024%20Crypto%20Crime%20Report.pdf?version=0/`).

Weili Chen, Xiaoqing Guo, Zibin Chen, Zongyou Zheng, and Yutong Lu. Phishing scam detection on ethereum: Towards financial security for blockchain ecosystem. *Proceedings of the 29th International Joint Conference on Artificial Intelligence*, pages 4506–4512, 2020a.

Zibin Chen, Jiajing Wu, Qing Yuan, Shaoxia Du, and Yanghua Zhou. E-gcn: An efficient graph neural network for ethereum phishing node detection. *2020 IEEE International Conference on Blockchain (Blockchain)*, pages 9–15, 2020b.

Mauro Conti, Ankit Gangwal, and Sushmita Ruj. A survey on security and privacy issues of bitcoin. *IEEE Communications Surveys Tutorials*, 20(4):3416–3452, 2018.

Mark Holub and Jackie O'Connor. What's new in crypto-land? exploring the noncriminal bitcoin network ecosystem. *2018 17th IEEE International Conference on Trust, Security and Privacy in Computing and Communications/12th IEEE International Conference on Big Data Science and Engineering (TrustCom/BigDataSE)*, pages 1407–1412, 2018.

Marco Iansiti and Karim R Lakhani. The truth about blockchain. *Harvard Business Review*, 95(1): 118–127, 2017.

Peng Li, Ting Li, Qiang Lin, Xiaofei Liu, and Xiaocheng Zhou. Pdgnn: An end-to-end phishing detection model based on graph neural network. *arXiv preprint arXiv:2201.04719*, 2022a.

Siyuan Li, Jiajing Wu, Jieli Liu, Shichang Zhang, and Qingshan Wang. Enhancing phishing scam detection in ethereum via temporal transaction aggregation graph network. *IEEE Transactions on Computational Social Systems*, 2022b.

Guanchen Lv, Zhenghang Li, Yini Wu, Long Xiang, and Yuan He. Graph embedding-based phishing detection in ethereum. *2021 International Conference on Blockchain and Trustworthy Systems (BlockSys)*, pages 170–181, 2021.

Nino Shervashidze, Pascal Schweitzer, Erik Jan Van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(9), 2011.

Qi Wang, Chao Liu, Wanrong Miao, Chenghua Li, and Xiaocheng Zhou. Tsgn: Transaction subgraph networks for ethereum phishing detection. *2021 IEEE International Conference on Big Data (Big Data)*, pages 792–797, 2021.

Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32, 2014.

Jiajing Wu, Jieli Liu, Yuzhou Zhong, Jie Zhang, and Zibin Zheng. Who are the phishers? phishing scam detection on ethereum via network embedding. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 52(2):1156–1166, 2020.

Yong Yuan and Fei-Yue Wang. Blockchain technology and its applications: A survey. *Journal of Industrial Information Integration*, 10:1–9, 2018.

Zibin Zheng, Shaoan Xie, Hong-Ning Dai, Xiangping Chen, and Huaimin Wang. An overview on smart contracts: Challenges, advances and platforms. *Future Generation Computer Systems*, 105: 475–491, 2020.