GraphSAGE Node Classification Report

Name: Taha Abdelwadoud

Student ID: 2205030

December 3, 2025

Abstract

This report describes the implementation and analysis of a GraphSAGE-based Graph Neural Network (GNN) for classifying nodes into benign and malicious categories. The experiment uses a small manually constructed graph of six nodes, with GraphSAGE performing neighbor aggregation to learn meaningful representations. The model is trained using PyTorch Geometric and achieves accurate predictions for all nodes.

Chapter 1

Introduction

Graph Neural Networks (GNNs) are modern machine learning models designed to operate directly on graph-structured data. They incorporate information from neighboring nodes to compute richer embeddings. GraphSAGE, in particular, is a sampling-based inductive GNN that aggregates information from a node's neighbors.

This experiment demonstrates how GraphSAGE can classify benign and malicious users in a small synthetic graph.

Chapter 2

Dataset Construction

We construct a graph with 6 nodes divided into two groups:

- Nodes 0, 1, 2 → benign users
- Nodes 3, 4, 5 → malicious users

Each node has a 2-dimensional feature vector:

- Benign: [1, 0]
- Malicious: [0, 1]

The graph contains:

- Fully connected benign subgraph (0–1–2)
- Fully connected malicious subgraph (3–4–5)
- One cross-connection between node 2 (benign) and node 3 (malicious)

Node and Edge Definitions (Code)

```
x = torch.tensor([
    [1.0, 0.0],
    [1.0, 0.0],
    [1.0, 0.0],
    [0.0, 1.0],
    [0.0, 1.0],
    [0.0, 1.0]
], dtype=torch.float)
```

```
edge_index = torch.tensor([
    [0,1],[1,0],[1,2],[2,1],[0,2],[2,0],
    [3,4],[4,3],[4,5],[5,4],[3,5],[5,3],
    [2,3],[3,2]
]).t().contiguous()

y = torch.tensor([0,0,0,1,1,1], dtype=torch.long)
```

The `edge_index` tensor is transposed to follow PyTorch Geometric's format: *shape = [2, num_edges]*.

Chapter 3

GraphSAGE Model

A two-layer GraphSAGE network is implemented. The architecture:

- Input layer: 2-dimensional features
- Hidden layer: 4-dimensional embedding
- Output layer: 2 classes (benign, malicious)

Model Definition (Code)

```
class GraphSAGENet(torch.nn.Module):
    def __init__(self, in_channels, hidden_channels, out_channels):
        super(GraphSAGENet, self).__init__()
        self.conv1 = SAGEConv(in_channels, hidden_channels)
        self.conv2 = SAGEConv(hidden_channels, out_channels)

    def forward(self, x, edge_index):
        x = self.conv1(x, edge_index)
        x = F.relu(x)
        x = self.conv2(x, edge_index)
        return F.log_softmax(x, dim=1)
```

Chapter 4

Training the Model

The model is trained for 50 epochs using:

- Loss: Negative Log-Likelihood (NLL)
- Optimizer: Adam (learning rate = 0.01)

Training Loop (Code)

```
optimizer = torch.optim.Adam(model.parameters(), lr=0.01)
model.train()

for epoch in range(50):
    optimizer.zero_grad()
    out = model(data.x, data.edge_index)
    loss = F.nll_loss(out, data.y)
    loss.backward()
    optimizer.step()
```

Chapter 5

Results

After training, the model correctly predicts all node labels:

```
Predicted labels: [0, 0, 0, 1, 1, 1]
```

This indicates that GraphSAGE successfully learned the relationship between node features and their local graph structure.

Interpretation

- Benign cluster (0–1–2) is consistently classified correctly.
- Malicious cluster (3–4–5) is also classified correctly.
- The cross-edge between node 2 and node 3 does not confuse the model.
- Graph structure + simple features were sufficient for perfect separation.

Chapter 6

Conclusion

The experiment demonstrates that GraphSAGE can effectively classify nodes even in a small graph with simple features. The method aggregates neighborhood information and learns discriminative embeddings for benign and malicious users.

This setup forms a foundation for more advanced cybersecurity node classification tasks such as:

- Botnet detection
- Insider threat modeling
- Social engineering behavior analysis
- Network anomaly detection