Classifying Malicious Nodes with GraphSAGE

Consider a toy social network of 6 users. We manually assign two "benign" user feature vectors and two "malicious" vectors, and connect them in a small graph. We then train a two-layer GraphSAGE to classify each node as benign (0) or malicious (1):

- We build a Data object for 6 nodes, with simple 2-dimensional features.
- Edges are set so that benign nodes (0,1,2) form a small clique and malicious nodes (3,4,5) form another, with one cross-link (2–3).
- The GraphSAGE model (GraphSAGENet) has two SAGEConv layers. Each layer automatically samples neighbors (here small graph, so it effectively uses all neighbors) and aggregates their features.
- During training, the model learns weights so that benign nodes output class 0 and malicious output class 1.

After training, the model should correctly classify all nodes (the printout shows the predicted labels). The learned node embeddings reflect each node's own features **plus** the averaged features of its neighbors. In practice, malicious nodes with predominantly malicious neighbors will have embeddings that the classifier sees as class 1, while benign nodes cluster around class 0.

1. Importing Required Libraries

imports all the essential libraries needed to build and train a GraphSAGE model.

- torch provides tensor operations and neural network components.
- torch_geometric.data.Data stores graph-structured data.
- SAGEConv implements the GraphSAGE convolution layers.
- torch.nn.functional provides activation functions and loss functions.

These imports prepare the environment for building a graph-based machine learning model.

---

2. Creating a Simple Graph with Node Features

construct a small graph with **6 nodes**, each having **2 input features**.
We use a simple representation:

- [1, 0] for benign users
- [0, 1] for malicious users

This feature encoding helps the model distinguish between the two classes during training.

---

3. Defining the Graph Structure (Edges)

The edge_index tensor describes all connections between nodes.
The graph has two communities:

- A benign cluster (nodes 0, 1, 2) which are fully connected

- A malicious cluster (nodes 3, 4, 5) which are fully connected
- One "bridge edge" connecting node 2 (benign) to node 3 (malicious)

This structure allows the model to learn how behavior spreads through different communities and how one malicious connection can affect neighborhood aggregation.

---

4. Assigning Labels

Each node is assigned a label:

- 0 benign
- 1 malicious

These labels are used during training so the model learns a supervised classification task.

---

5. Building the GraphSAGE Neural Network

define a **two-layer GraphSAGE model**:

- **First GraphSAGE layer:** learns local neighborhood patterns and produces intermediate node embeddings.
- **ReLU activation:** introduces non-linearity to improve learning.
- **Second GraphSAGE layer:** generates final node representations used for classification.
- **Log-Softmax:** outputs log-probabilities for each class (benign/malicious).

This architecture allows the model to aggregate information from each node's neighbors and classify them based on structural and feature signals.

---

6. Initializing the Model

- in_channels = 2 (the two input features per node)
- hidden_channels = 4 (embedding size after the first layer)
- out_channels = 2 (number of classes: benign vs malicious)

This sets up the model for binary classification on our custom graph.

---

7. Training the Model

The training loop runs for 50 epochs:

1. Compute predictions for all nodes
2. Calculate the loss using **Negative Log-Likelihood (NLL)**
3. Backpropagate the error
4. Update model weights using the Adam optimizer

The model gradually learns how to classify nodes based on their features and graph connections.

---

8. Evaluating the Model

After training, we switch the model to evaluation mode.
We compute predictions for all nodes and select the class with the highest probability.

Ideally, the predicted labels should match the true labels:
[0, 0, 0, 1, 1, 1]

This indicates the model successfully learned to distinguish between benign and malicious nodes.
