Classifying Nodes in a Social Graph Using GraphSAGE

Course: Social Networks

Name: Mohamed Hassan Ahmed
ID: 2205200

Define Node Features

```
x = torch.tensor(
  [
    [1.0, 0.0],  # Node 0 (benign)
    [1.0, 0.0],  # Node 1 (benign)
    [1.0, 0.0],  # Node 2 (benign)
    [0.0, 1.0],  # Node 3 (malicious)
    [0.0, 1.0],  # Node 4 (malicious)
    [0.0, 1.0]  # Node 5 (malicious)
  ],
  dtype=torch.float,
)
```

Explanation

This block creates a feature matrix for all six nodes in the graph.
Each node is represented by two features:

[1, 0] → represents a benign user
[0, 1] → represents a malicious user

The features are stored as a PyTorch tensor of type float.

This matrix x will be used as input to the GraphSAGE model, allowing it to learn patterns based on node attributes as well as graph structure.

Define Graph Connections (Edges)

```
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],  # one connection between a benign (2) and malicious (3)
    ],
    dtype=torch.long,
  )
  .t()
  .contiguous()
)
```

Explanation

This block defines the edges of the graph, indicating which nodes are connected.
edge_index uses COO (coordinate) format: each pair [i, j] represents a directed edge from node i to node j.
Since the graph is undirected, each edge is written twice (both directions).

Graph structure:
Benign cluster: Nodes 0, 1, 2 are fully connected.
Malicious cluster: Nodes 3, 4, 5 are fully connected.
Cross-community link: Node 2 (benign) is connected to node 3 (malicious).

The .t() transposes the tensor to match PyTorch Geometric's expected shape (2, num_edges).
.contiguous() ensures memory is stored in a continuous block for efficiency.

This setup allows the GraphSAGE model to propagate information along the edges, letting each node aggregate features from its neighbors.

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

data = Data(x=x, edge_index=edge_index, y=y)

Define Node Labels and Create Data Object

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

data = Data(x=x, edge_index=edge_index, y=y)

Explanation

y contains the ground-truth labels for all nodes:

0 → benign nodes (0, 1, 2)

1 → malicious nodes (3, 4, 5)

The Data object combines:

x → node features

edge_index → graph connections

y → node labels

This structure is standard in PyTorch Geometric and makes it easy to pass the graph into a GNN model.

Define GraphSAGE Model

```
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):
    # First layer: sample neighbors and aggregate
    x = self.conv1(x, edge_index)
    x = F.relu(x)  # non-linear activation
    # Second layer: produce final embeddings/class scores
    x = self.conv2(x, edge_index)
    return F.log_softmax(x, dim=1)  # log-probabilities for classes
```

Explanation

This defines a two-layer GraphSAGE neural network.

Layer details:
First GraphSAGE layer (conv1)
Input dimension = 2 (each node has 2 features)

Hidden dimension = 4
Aggregates features from neighbors to produce new node embeddings
Followed by ReLU activation to introduce non-linearity
Second GraphSAGE layer (conv2)
Input dimension = 4 (from hidden layer)
Output dimension = 2 (scores for benign vs malicious)
Produces final node embeddings and class scores
F.log_softmax converts the output to log-probabilities, which works with the negative log-likelihood loss during training.
Purpose:
GraphSAGE combines node features and neighbor information to learn meaningful embeddings for classification. Each node's final representation reflects both its own features and its local graph structure.

Instantiate the Model

model = GraphSAGENet(in_channels=2, hidden_channels=4, out_channels=2)

Explanation

Creates an instance of the GraphSAGE network defined earlier.

Parameters:
in_channels=2 → each node has 2 input features
hidden_channels=4 → hidden layer dimension
out_channels=2 → two output classes: benign (0) and malicious (1)

Set Up Optimizer and Training Loop

The model is now ready to be trained on the graph data.

```
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)  # negative log-likelihood
  loss.backward()
  optimizer.step()
```

Explanation

Optimizer
Adam optimizer updates the model parameters using the computed gradients.
Learning rate = 0.01.
Training Mode
model.train() enables training behaviors (e.g., dropout if used).
Epoch Loop
Loops 50 times over the same data (because the graph is very small).
Forward Pass
out = model(data.x, data.edge_index)
Computes predictions (log-probabilities) for all nodes.
Compute Loss
F.nll_loss(out, data.y) calculates negative log-likelihood loss comparing predictions to true labels.
Backpropagation
loss.backward() computes gradients.
optimizer.step() updates the model weights.
Purpose:
This loop trains the GraphSAGE model to classify nodes correctly using both their features and graph connectivity. Because the graph is small and the node features are clear, the model quickly learns to separate benign and malicious nodes.

Model Evaluation and Predictions

```
model.eval()
pred = model(data.x, data.edge_index).argmax(dim=1)
print("Predicted labels:", pred.tolist())
```

Explanation

Switch to evaluation mode
model.eval() disables training-specific behaviors (like dropout or batch normalization updates).

Ensures predictions are stable.

Generate predictions

model(data.x, data.edge_index) runs a forward pass on the trained model.

.argmax(dim=1) selects the class with the highest probability for each node.

Result is a 1D tensor containing predicted labels for all nodes.

Output predictions

pred.tolist() converts the tensor to a standard Python list for easy display.

Example output: [0, 0, 0, 1, 1, 1]

Interpretation:

Nodes 0–2 are predicted as benign (0)

Nodes 3–5 are predicted as malicious (1)

This shows that the model successfully learned the distinction between the two groups based on features and graph connections.

Prediction Results

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

interpretation

Nodes 0, 1, 2 → predicted as benign (0)
Nodes 3, 4, 5 → predicted as malicious (1)

- ✧ This means the model correctly classified all nodes, achieving perfect accuracy on this small synthetic graph.

Why it worked:

- ✧ Node features clearly distinguish benign vs malicious users ([1,0] vs [0,1]).
- ✧ Graph structure shows two separate communities with only a single cross-edge, making the pattern easy for GraphSAGE to learn.
- ✧ The small size of the graph allows the model to converge quickly during training.