Coding Exercise: Implement a Node Classification Model using Graph Neural Networks in PyTorch Geometric

Problem Statement

You are given a citation network where nodes represent papers, and edges represent citations between them. Each node has a feature vector representing the paper's text embedding, and each node is labeled with a research category. Your task is to build a **Graph Neural Network (GNN) for node classification** using **PyTorch Geometric**.

Dataset

Use the built-in **Cora dataset** from PyTorch Geometric:

python
CopyEdit
```
from torch_geometric.datasets import Planetoid
dataset = Planetoid(root='data', name='Cora')
```

This dataset contains:

- **Nodes**: Research papers
- **Edges**: Citation relationships
- **Features**: Bag-of-words representation of the paper
- **Labels**: Research field (7 classes)

Task Requirements

1. **Build a Graph Neural Network (GNN)** using PyTorch Geometric that classifies nodes into one of the 7 categories.
2. Use a **Graph Convolutional Network (GCN)** or **GraphSAGE** as the backbone model.
3. Implement **training and evaluation loops** with PyTorch.
4. Achieve at least **70% accuracy** on the test set.
5. Ensure proper **GPU utilization** if available.

Starter Code

Below is a skeleton to guide your implementation:

python
CopyEdit
```
import torch
import torch.nn.functional as F
```

```python
from torch_geometric.nn import GCNConv
from torch_geometric.datasets import Planetoid
from torch_geometric.loader import DataLoader

# Load dataset
dataset = Planetoid(root='data', name='Cora')

# Define a simple GCN model
class GCN(torch.nn.Module):
    def __init__(self, input_dim, hidden_dim, output_dim):
        super(GCN, self).__init__()
        self.conv1 = GCNConv(input_dim, hidden_dim)
        self.conv2 = GCNConv(hidden_dim, output_dim)

    def forward(self, data):
        x, edge_index = data.x, data.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)

# Training loop
def train(model, data, optimizer, epochs=200):
    model.train()
    for epoch in range(epochs):
        optimizer.zero_grad()
        out = model(data)
        loss = F.nll_loss(out[data.train_mask],
data.y[data.train_mask])
        loss.backward()
        optimizer.step()
        if epoch % 10 == 0:
            print(f'Epoch {epoch}, Loss: {loss.item():.4f}')

# Evaluate function
def evaluate(model, data):
    model.eval()
    _, pred = model(data).max(dim=1)
    correct = (pred[data.test_mask] == data.y[data.test_mask]).sum()
    acc = correct.item() / data.test_mask.sum().item()
    print(f'Test Accuracy: {acc:.4f}')

# Instantiate and train the model
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
data = dataset[0].to(device)
model = GCN(dataset.num_features, 16, dataset.num_classes).to(device)
optimizer = torch.optim.Adam(model.parameters(), lr=0.01,
weight_decay=5e-4)

train(model, data, optimizer)
evaluate(model, data)
```

Evaluation Criteria

- **Correct implementation of the GNN model** (GCN or GraphSAGE).
- **Proper use of PyTorch Geometric** for handling graph data.
- **Effective training and evaluation loops**.
- **Efficient use of GPU** when available.
- **Interpretability**: Code should be clean and well-structured.

Bonus Challenges (Optional)

1. Modify the model to use **GraphSAGE** instead of GCN.
2. Add **dropout layers** to improve regularization.
3. Implement an **early stopping** mechanism based on validation loss.