import torch

import torch.nn as nn

import torch.optim as optim

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from torch.utils.data import DataLoader, TensorDataset

from sklearn.metrics import f1\_score, recall\_score

import random

import numpy as np

import csv

# 定义MLP模型

class MLP(nn.Module):

def \_\_init\_\_(self):

super(MLP, self).\_\_init\_\_()

self.fc1 = nn.Linear(8, 128)

self.fc2 = nn.Linear(128, 64)

self.output = nn.Linear(64, 1)

self.relu = nn.ReLU()

self.sigmoid = nn.Sigmoid()

def forward(self, x):

x = self.relu(self.fc1(x))

x = self.relu(self.fc2(x))

x = self.sigmoid(self.output(x))

return x

# 加载数据

def load\_data\_and\_export\_scaler\_params(filename):

data = pd.read\_csv(filename, encoding='GBK')

X = data.iloc[:, :-1].values

y = data.iloc[:, -1].values.reshape(-1, 1)

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# 获取均值和标准差

means = scaler.mean\_

stds = scaler.scale\_

# 将均值和标准差保存到CSV文件

params\_df = pd.DataFrame({'mean': means, 'std': stds})

params\_df.to\_csv('F:\中南大学系列\论文撰写\Multi-fidelity Neural Network\长春数据\/1.低保真模型（基于长春数据训练的）/means\_and\_stds.csv', encoding='GBK', index=False)

return X\_scaled, y

# 数据集划分 按训练集：验证集：测试集 = 6 ：2 ：2的比例

def split\_data(X, y):

X\_train, X\_temp, y\_train, y\_temp = train\_test\_split(X, y, test\_size=0.4, random\_state=42)

X\_val, X\_test, y\_val, y\_test = train\_test\_split(X\_temp, y\_temp, test\_size=0.5, random\_state=42)

# 保存X\_train, y\_train

pd.DataFrame(X\_train).to\_csv('F:\中南大学系列\论文撰写\Multi-fidelity Neural Network\长春数据\/1.低保真模型（基于长春数据训练的）\划分的数据集\X\_train.csv', encoding='GBK',index=False)

pd.DataFrame(y\_train).to\_csv('F:\中南大学系列\论文撰写\Multi-fidelity Neural Network\长春数据\/1.低保真模型（基于长春数据训练的）\划分的数据集\y\_train.csv', encoding='GBK',index=False)

# 保存X\_val, y\_val

pd.DataFrame(X\_val).to\_csv('F:\中南大学系列\论文撰写\Multi-fidelity Neural Network\长春数据\/1.低保真模型（基于长春数据训练的）\划分的数据集\X\_val.csv', encoding='GBK',index=False)

pd.DataFrame(y\_val).to\_csv('F:\中南大学系列\论文撰写\Multi-fidelity Neural Network\长春数据\/1.低保真模型（基于长春数据训练的）\划分的数据集\y\_val.csv', encoding='GBK',index=False)

# 保存X\_test, y\_test

pd.DataFrame(X\_test).to\_csv('F:\中南大学系列\论文撰写\Multi-fidelity Neural Network\长春数据\/1.低保真模型（基于长春数据训练的）\划分的数据集\X\_test.csv', encoding='GBK',index=False)

pd.DataFrame(y\_test).to\_csv('F:\中南大学系列\论文撰写\Multi-fidelity Neural Network\长春数据\/1.低保真模型（基于长春数据训练的）\划分的数据集\y\_test.csv', encoding='GBK',index=False)

return X\_train, X\_val, X\_test, y\_train, y\_val, y\_test

# 创建数据加载器

def create\_dataloader(X, y, batch\_size=32):

tensor\_x = torch.Tensor(X)

tensor\_y = torch.Tensor(y)

dataset = TensorDataset(tensor\_x, tensor\_y)

dataloader = DataLoader(dataset, batch\_size=batch\_size, shuffle=True)

return dataloader

# 训练模型并记录损失

def train\_model(model, train\_loader, val\_loader, criterion, optimizer, num\_epochs=100):

train\_losses = []

val\_losses = []

for epoch in range(num\_epochs):

model.train()

total\_train\_loss = 0

for inputs, labels in train\_loader:

optimizer.zero\_grad()

outputs = model(inputs)

loss = criterion(outputs, labels)

loss.backward()

optimizer.step()

total\_train\_loss += loss.item()

avg\_train\_loss = total\_train\_loss / len(train\_loader)

train\_losses.append(avg\_train\_loss)

model.eval()

total\_val\_loss = 0

with torch.no\_grad():

for inputs, labels in val\_loader:

outputs = model(inputs)

loss = criterion(outputs, labels)

total\_val\_loss += loss.item()

avg\_val\_loss = total\_val\_loss / len(val\_loader)

val\_losses.append(avg\_val\_loss)

print(f'Epoch {epoch + 1}/{num\_epochs}, Training Loss: {avg\_train\_loss}, Validation Loss: {avg\_val\_loss}')

# 创建DataFrame并保存到CSV

df = pd.DataFrame({

'Epoch': range(1, num\_epochs + 1),

'Training Loss': train\_losses,

'Validation Loss': val\_losses

})

df.to\_csv('F:\中南大学系列\论文撰写\Multi-fidelity Neural Network\长春数据\/1.低保真模型（基于长春数据训练的）\/training\_validation\_loss(构造的数据).csv', index=False)

print("Losses were saved to training\_validation\_loss.csv")

return train\_losses, val\_losses

# # 测试模型并计算F1分数和召回率

def model\_test(model, test\_loader, threshold=0.5):

model.eval()

all\_labels = []

all\_predictions = []

results = []

output\_csv\_path = 'F:\中南大学系列\论文撰写\Multi-fidelity Neural Network\长春数据\/1.低保真模型（基于长春数据训练的）\测试集预测结果\predictions.csv'

with torch.no\_grad():

for inputs, labels in test\_loader:

outputs = model(inputs)

predicted = (outputs >= threshold).float() # 使用0.5作为分类阈值

all\_labels.extend(labels.view(-1).tolist())

all\_predictions.extend(predicted.view(-1).tolist())

# 记录每个样本的预测结果

for label, prediction in zip(labels.view(-1).tolist(), predicted.view(-1).tolist()):

correct = 'yes' if label == prediction else 'no'

results.append((label, prediction, correct))

# 计算整体的F1分数

f1 = f1\_score(all\_labels, all\_predictions)

# 计算每个类别的召回率

recalls = recall\_score(all\_labels, all\_predictions, average=None)

# 输出F1分数和每个类别的召回率

print(f'Overall F1 Score on test set: {f1:.2f}')

print('Recall for each class:', {f'Class {i}': r for i, r in enumerate(recalls, start=0)})

# 保存预测标签到CSV文件

with open(output\_csv\_path, mode='w', newline='') as file:

writer = csv.writer(file)

writer.writerow(['Actual', 'Predicted', 'Correct']) # 写入表头

for actual, predicted, correct in results:

writer.writerow([actual, predicted, correct])

print(f'Predictions have been saved to {output\_csv\_path}')

# 假设已经有了model和test\_loader

# model\_test(model, test\_loader)

# 保存模型

def save\_model(model, path):

torch.save(model.state\_dict(), path)

# 加载模型

def load\_model(model, path):

model.load\_state\_dict(torch.load(path))

model.eval()

return model

# 绘制损失曲线

def plot\_losses(train\_losses, val\_losses):

plt.figure(figsize=(10, 5))

plt.plot(train\_losses, label='Training Loss')

plt.plot(val\_losses, label='Validation Loss')

plt.title('Loss Curve')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.legend()

plt.grid(True)

plt.savefig('F:\中南大学系列\论文撰写\迁移学习+泥饼过程预测\数据\出图\训练损失曲线/loss\_curve（图标知识模型）.png', dpi=1200, bbox\_inches='tight') # 保存为PNG文件，分辨率1200 DPI

plt.show()

plt.close() # 关闭图形，防止重复显示

# 实际使用示例中包含测试部分

X, y = load\_data\_and\_export\_scaler\_params('F:\中南大学系列\论文撰写\Multi-fidelity Neural Network\长春数据\/0.1最终构造的低保真数据\低保真数据.csv') # 替换为实际的CSV文件路径

X\_train, X\_val, X\_test, y\_train, y\_val, y\_test = split\_data(X, y)

train\_loader = create\_dataloader(X\_train, y\_train)

val\_loader = create\_dataloader(X\_val, y\_val)

test\_loader = create\_dataloader(X\_test, y\_test)

model = MLP()

criterion = nn.BCELoss()

optimizer = optim.Adam(model.parameters(), lr=0.001)

train\_losses, val\_losses = train\_model(model, train\_loader, val\_loader, criterion, optimizer)

# plot\_losses(train\_losses, val\_losses)

model\_test(model, test\_loader)

# 模型保存路径

model\_save\_path = 'F:\中南大学系列\论文撰写\Multi-fidelity Neural Network\长春数据\/1.低保真模型（基于长春数据训练的）\模型保存/mlp\_model(Universal classification diagram).model' #事实上，保存模型时，是否使用后缀没有什么影响

save\_model(model, model\_save\_path)

# # 加载模型示例（如果需要）

# loaded\_model = load\_model(MLP(), model\_save\_path)