多层感知机实践

源代码

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import numpy as np

# 导入数据
df = pd.read_csv(r"C:\Users\30975\Downloads\dataset.csv")
# print(df)

# 数据预处理
# print(df.isnull().sum())  # 经检验无缺失值，故略去这一步

# 使用 StandardScaler 进行标准化
scaler = StandardScaler()
columns_standardize = ['id','Marital status','Application mode','Application
order','Course','Daytime/evening attendance']
df_standardized = df.copy()
df_standardized[columns_standardize] =
scaler.fit_transform(df[columns_standardize])

# 选择特征列
features = ['id','Marital status','Application mode','Application
order','Course','Daytime/evening attendance']
X = df[features].values
y = df['Target'].values

# 划分训练集和测试集
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# 数据标准化
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# 创建mlp模型
class MLP(nn.Module):
    def __init__(self, input_layer, hidden_layer, output_layer):
        super(MLP, self).__init__()
        self.fc1 = nn.Linear(input_layer, hidden_layer)  # 第一个全连接层
        self.fc2 = nn.Linear(hidden_layer, hidden_layer)  # 第二个全连接层
        self.fc3 = nn.Linear(hidden_layer, output_layer)  # 输出层

    def forward(self, x):
        x = F.relu(self.fc1(x))  # 应用 ReLU 激活函数
```

```
        x = F.relu(self.fc2(x))  # 应用 ReLU 激活函数
        x = self.fc3(x)  # 输出层
        return x

device = torch.device("cpu")

input_dim = len(features)  # 输入维度等于特征数量
hidden_dim = 64  # 隐藏层的节点数
output_dim = len(np.unique(y))  # 输出维度，等于目标类别的数量
model = MLP(input_dim, hidden_dim, output_dim).to(device)

criterion = nn.CrossEntropyLoss().to(device)
optimizer = optim.Adam(model.parameters(), lr=0.01)

# 将 NumPy 数组转换为 PyTorch 张量
train_data = torch.from_numpy(X_train).float()
train_labels = torch.from_numpy(y_train).long()
test_data = torch.from_numpy(X_test).float()
test_labels = torch.from_numpy(y_test).long()

# 定义数据加载器
train_dataset = torch.utils.data.TensorDataset(train_data, train_labels)
train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=32,
shuffle=True)

test_dataset = torch.utils.data.TensorDataset(test_data, test_labels)
test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=32,
shuffle=False)

# 训练函数
def train(model, device, train_loader, optimizer, criterion, epoch):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = criterion(output, target)
        loss.backward()
        optimizer.step()
        if batch_idx % 100 == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                epoch, batch_idx * len(data), len(train_loader.dataset),
                100. * batch_idx / len(train_loader), loss.item()))

# 测试函数
def test(model, device, test_loader, criterion):
    model.eval()
    test_loss = 0
    correct = 0
    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(device), target.to(device)
            output = model(data)
            test_loss += criterion(output, target).item()
            pred = output.argmax(dim=1, keepdim=True)
            correct += pred.eq(target.view_as(pred)).sum().item()
```

```
    test_loss /= len(test_loader.dataset)
    print('\nTest set: Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)\n'.format(
        test_loss, correct, len(test_loader.dataset),
        100. * correct / len(test_loader.dataset)))

# 开始训练
num_epochs = 10
for epoch in range(1, num_epochs + 1):
    train(model, device, train_loader, optimizer, criterion, epoch)
    test(model, device, test_loader, criterion)
```

问题与解决

1. 首先在我最开始尝试自己编码时，我遇到的第一个问题的变量的命名，一整篇代码下来需要非常大量的变量，我最开始的习惯就是设置为比如 a1，a2，a3……等等，但后来我发现经常可能今天明白这个变量代表什么，但是明天再开始时就忘了，尤其是需要更改一些内容时，极其头疼，但是有一次使用了AI写了一次代码后，我发现了AI命名变量的方法特别实用，以前不明白下划线为什么要用在变量命名里面，现在才领悟到有下划线作为一个小分割，可以使变量的命名清晰而实用
2. 第二点就是没能区分好标准化和归一化，认为二者都要同时使用，但其实一般来说二者取其一即可

   区别标准化与归一化

   - **标准化**：目的是将数据转换成“标准正态分布”的形式，即均值为0，标准差为1

     适用于数据没有明确边界下，且符合高斯分布时
   - **归一化**：目的是将数据等比例缩放，使之落入一个特定的区间，通常时0~1