homework1

2025 年 3 月 29 日

[1]:
```
import numpy as np
import pandas as pd
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error

# 1. 数据读取
def load_data(excel_path):
    # 读取 Excel 数据（需替换实际文件路径）
    df = pd.read_excel(excel_path, header=None, usecols="A:N", skiprows=1,␣
 ↪nrows=506)
    data = df.values.astype(np.float32)
    return data
data = load_data('C:/Users/ustud/Downloads/BostonHousingData.xlsx')

# 2. 数据预处理
X_train_np = data[:450, :13]  # 前 450 条训练数据
y_train_np = data[:450, 13]   # 目标变量
X_test_np = data[450:, :13]   # 后 56 条测试数据
y_test_np = data[450:, 13]

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_np)
X_test_scaled = scaler.transform(X_test_np)
```

```
X_train = torch.tensor(X_train_scaled, dtype=torch.float32)
y_train = torch.tensor(y_train_np, dtype=torch.float32).unsqueeze(1)
X_test = torch.tensor(X_test_scaled, dtype=torch.float32)
y_test = torch.tensor(y_test_np, dtype=torch.float32).unsqueeze(1)

# 3. 创建数据加载器
train_dataset = TensorDataset(X_train, y_train)
test_dataset = TensorDataset(X_test, y_test)

batch_size = 32
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)

# 4. 定义神经网络模型
class RegressionModel(nn.Module):
    def __init__(self):
        super(RegressionModel, self).__init__()
        self.net = nn.Sequential(
            nn.Linear(13, 64),
            nn.ReLU(),
            nn.Dropout(0.2),
            nn.Linear(64, 32),
            nn.ReLU(),
            nn.Dropout(0.2),
            nn.Linear(32, 1)
        )

    def forward(self, x):
        return self.net(x)

# 5. 模型初始化
model = RegressionModel()
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# 6. 训练过程
num_epochs = 200
```

```python
for epoch in range(num_epochs):
    model.train()
    epoch_loss = 0.0

    for inputs, targets in train_loader:
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, targets)
        loss.backward()
        optimizer.step()
        epoch_loss += loss.item() * inputs.size(0)

    avg_loss = epoch_loss / len(train_loader.dataset)

    # 每 20 个 epoch 打印进度
    if (epoch+1) % 20 == 0:
        print(f"Epoch [{epoch+1}/{num_epochs}], 训练损失: {avg_loss:.4f}")

# 7. 模型评估
model.eval()
with torch.no_grad():
    test_predictions = model(X_test)
    mse = criterion(test_predictions, y_test)
    print("\n模型评估结果: ")
    print(f"测试集均方误差 (MSE): {mse.item():.4f}")

    # 样例输出对比
    print("\n前 5 个样本预测对比: ")
    print("真实值\t预测值")
    for i in range(5):
        print(f"{y_test[i].item():.4f} | {test_predictions[i].item():.4f}")
```

```
Epoch [20/200], 训练损失: 35.1966
Epoch [40/200], 训练损失: 27.1562
Epoch [60/200], 训练损失: 27.2511
Epoch [80/200], 训练损失: 23.5428
```

Epoch [100/200], 训练损失: 22.6225
Epoch [120/200], 训练损失: 21.4608
Epoch [140/200], 训练损失: 22.1250
Epoch [160/200], 训练损失: 18.4468
Epoch [180/200], 训练损失: 19.4141
Epoch [200/200], 训练损失: 20.9988

模型评估结果:
测试集均方误差 (MSE): 16.4418

前 5 个样本预测对比:
真实值　　预测值
13.4000 | 13.2264
15.2000 | 15.9849
16.1000 | 16.2923
17.8000 | 16.4677
14.9000 | 11.8478