中国科学技术大学　微电子学院

《神经网络及其应用》实验报告一

姓名　杨翊麟　　**学号**　BC24219010

时间　2025/03/23　　**指导教师**　徐奇

实验名称：Regression Analysis 波士顿房价预测

一、实验目的

1、熟悉 Regression（回归）算法的构成及实现方式；
2、掌握 Python + PyTorch 编程环境下对波士顿房价预测数据集的回归分析；
3、构建神经网络，并使用测试集对训练好的神经网络模型进行测试；
4、计算波士顿房价的预测值与实际值之间的均方误差，评估模型的性能。

二、实验步骤

1、**数据准备：**读取 BostonHousingData.xlsx 数据集，进行**相关性分析**，选出与房价（MEDV）具有较高相关性的特征作为主要特征，绘制出相关性热图，并将前 450 条作为训练集，后 50 条作为测试集。
2、**数据预处理：**使用 **Min-Max 归一化**处理部分跨度较大的特征值。
3、**神经网络构建：**导入处理后的数据，使用 **PyTorch 框架**构建适合回归任务的神经网络模型（输入层、两层隐藏层、输出层）第一层隐藏层： 64 个神经元 ＋ReLU＋Dropout　第二层隐藏层： 32 个神经元 ＋ReLU＋Dropout，输出层：1 个神经元（预测值））
4、**模型训练：**使用训练集对神经网络模型进行多轮次训练（200 epochs），通过反向传播算法更新权重和偏置。
5、**模型测试和评估：**使用测试集对训练好的神经网络模型进行测试，计算预测值与实际值之间的均方误差作为评价指标，生成可视化折线图观察分析模型的性能。

三、实验流程分析

（1）库文件

torch.nn 用于构建神经网络；torch.optim 提供了优化器（如 Adam、SGD），用于更新神经网络偏重。NumPy 用于数据处理，如数组运算、归一化等。Matplotlib 是 Python 的绘图库，pyplot 这里用于绘制数据对比图、损失曲线等。Pandas 是数据分析库，这里用于读取 BostonHousingData.xlsx 数据。Seaborn 是数据可视化库，这里用于绘制特征相关性的热图，帮助分析特征

对房价的影响。

```
import torch
import torch.nn as nn
import torch.optim as optim
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
```

（2）数据准备与预处理

通过相关性分析，选取与房价具有较高相关性的特征作为主要特征，并生成热力图（图 1）。将波士顿房价数据集的前 450 条作为训练集，后 50 条作为测试集，通过 Min-Max 归一化对特征值预处理。

```
# 进行相关性分析，选择相关性较高的特征，减少过拟合
def select_features_by_correlation(data, target_column='MEDV',
threshold=0.4):
    """ 选择与目标变量相关性较高的特征 """
    df = pd.DataFrame(data, columns=['CRIM', 'ZN', 'INDUS', 'CHAS',
'NOX', 'RM', 'AGE',
                                     'DIS', 'RAD', 'TAX', 'PTRATIO', 'B',
'LSTAT', 'MEDV'])
    corr_matrix = df.corr()  # 计算相关性矩阵
    target_corr =
corr_matrix[target_column].abs().sort_values(ascending=False)  # 目
标变量相关性排序

    # 选取高于阈值的特征
    selected_features = target_corr[target_corr >
threshold].index.tolist()
    if target_column in selected_features:
        selected_features.remove(target_column)  # 移除目标列本身

    print("Selected Features:", selected_features)

    # 绘制相关性热图
    plt.figure(figsize=(10, 6))
    sns.heatmap(df[selected_features + [target_column]].corr(),
annot=True, cmap='coolwarm', fmt=".2f")
    plt.title("Feature Correlation Heatmap")
    plt.show()

    return selected_features
```

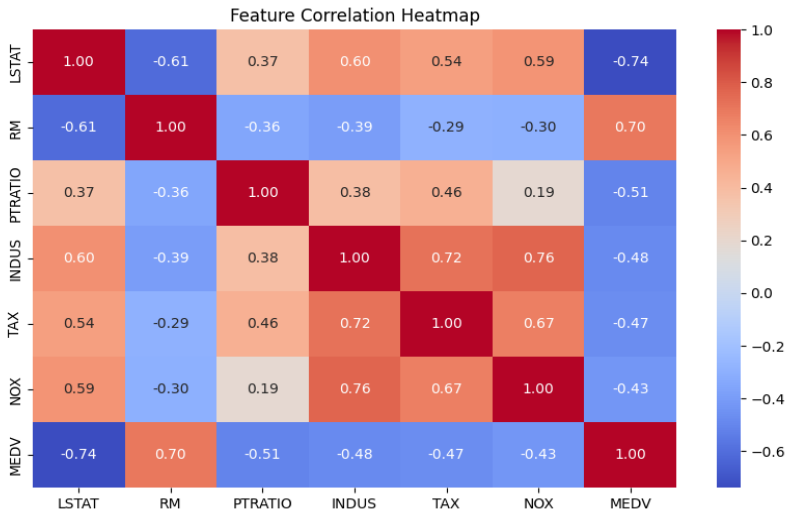

图 1 特征相关性热力图

```
# 加载并预处理数据
def load_data():
    datafile = 'BostonHousingData.xlsx'
    df = pd.read_excel(datafile, dtype=np.float32)
    data = df.to_numpy()

    if data.shape[0] != 506:
        raise ValueError(f"Dataset should have 506 rows, but found
{data.shape[0]}.")

    # 相关性分析
    selected_features = select_features_by_correlation(data)
    selected_indices = [df.columns.get_loc(f) for f in
selected_features]  # 获取索引

    # 仅选择相关性较高的特征
    data = data[:, selected_indices + [-1]]  # 选取主要特征 + 房价
（MEDV）

    # 使用前450条作为训练集
    train_data = data[:450]

    # Min-Max归一化对特征值预处理（基于训练集的最大/最小值）
    max_values = train_data.max(axis=0)
    min_values = train_data.min(axis=0)

    for i in range(data.shape[1]):
        if max_values[i] == min_values[i]:
```

```
            raise ValueError(f"Feature {i} has identical min and max
values, causing division by zero.")
        data[:, i] = (data[:, i] - min_values[i]) / (max_values[i] -
min_values[i])

    # 归一化后再分配数据
    train_data = data[:450]     # 训练集：前450条
    test_data = data[456:506]   # 测试集：取最后50条

    return    train_data,    test_data,    max_values,    min_values,
selected_features
```

（3）神经网络构建

构建的神经网络包含（输入层、隐藏层和输出层）（第一层隐藏层（输入层到隐藏层），64 个神经元 + ReLU + Dropout，第二层隐藏层，32 个神经元 + ReLU + Dropout），以及输出层，1 个神经元（用于预测房价）。由于是回归问题，所以输出层无需使用激活函数。

```
# 定义神经网络模型
class Regressor(nn.Module):
    def __init__(self, input_size):
        super(Regressor, self).__init__()
        self.model = nn.Sequential(
            nn.Linear(input_size, 64),  # 第一层隐藏层，64个神经元
            nn.ReLU(),  # 激活函数
            nn.Dropout(0.2),  # 添加 Dropout (20%)

            nn.Linear(64, 32),  # 第二层隐藏层，32个神经元
            nn.ReLU(),  # 激活函数
            nn.Dropout(0.2),  # 添加 Dropout (20%)

            nn.Linear(32, 1)  # 输出层
        )

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

    def fit_model(self, train_data, num_epochs, batch_size):
        criterion = nn.MSELoss()
        optimizer = optim.Adam(self.parameters(), lr=0.0005)  # 使用
Adam优化器，基于计算出的梯度更新神经网络的权重和偏置。学习率设为0.0005

        for epoch in range(num_epochs):
            mini_batches = [train_data[k:k + batch_size] for k in
```

```
range(0, len(train_data), batch_size)]
            for mini_batch in mini_batches:
                x = torch.tensor(mini_batch[:, :-1],
dtype=torch.float32)
                y = torch.tensor(mini_batch[:, -1:],
dtype=torch.float32)
                outputs = self(x)                # 前向传播（计算预测值）
                loss = criterion(outputs, y)     # 计算损失（使用均方误差
（MSELoss）计算回归任务的损失函数）
                optimizer.zero_grad()            # 清空梯度（由于PyTorch 默
认梯度是累积的，所以每次迭代前都需要清除上一次的梯度）
                loss.backward()                  # 反向传播（计算梯度）
                optimizer.step()                 # 更新参数（通过梯度下降来更
新）

            if (epoch + 1) % 10 == 0:
                print(f'Epoch [{epoch + 1}/{num_epochs}], Loss:
{loss.item():.4f}')
```

（4）模型训练与性能评估

在模型训练中，通过**均方误差**（**MSELoss**）计算回归的损失函数，并通过**反向传播算法**更新权重和偏置。为了使学习效果显著，学习率设为 0.0005，同时训练轮数设为 200 轮。在训练完成后，对原始数据与可视化数据的对比进行了可视化，生成了二者对比的折线图（图 2），图像显示训练效果较好，预测值与真实值十分接近。

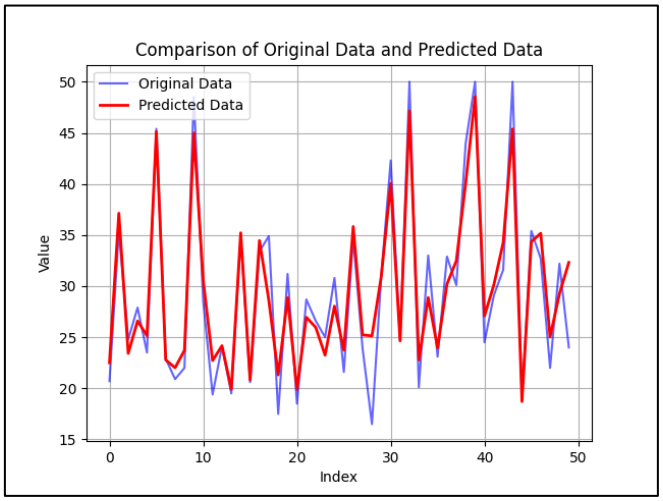

图 2 波士顿房价的原始值与预测值对比图

```
# 加载数据并训练模型
train_data, test_data, max_values, min_values,
selected_features = load_data()
model = Regressor(len(selected_features))  # 只使用挑选出的特征
model.fit_model(train_data, num_epochs=200, batch_size=16)   #
```

```
训练相对充分的轮次（200轮），使学习效果显著
torch.save(model.state_dict(), 'boston_model_nn.pth')
print("Model saved to boston_model_nn.pth")

# 可视化原始数据与预测数据的对比
def show_plt(origin, predict):
    plt.plot(origin, color='blue', label='Original Data',
alpha=0.6)
    plt.plot(predict, color='red', label='Predicted Data',
linewidth=2)
    plt.title('Comparison of Original Data and Predicted Data')
    plt.xlabel('Index')
    plt.ylabel('Value')
    plt.legend()
    plt.grid(True)
    plt.show()

# 进行预测并可视化
def predict_pytorch():
    loaded_model = Regressor(len(selected_features))

loaded_model.load_state_dict(torch.load('boston_model_nn.pth'))
    loaded_model.eval()  # 设置为评估模式（禁用 Dropout）
    x_test = torch.tensor(test_data[:, :-1],
dtype=torch.float32)
    y_true = test_data[:, -1]
    with torch.no_grad():
        y_pred = loaded_model(x_test).numpy().flatten()
    # 反归一化处理
    y_pred = y_pred * (max_values[-1] - min_values[-1]) +
min_values[-1]
    y_true = y_true * (max_values[-1] - min_values[-1]) +
min_values[-1]
    show_plt(y_true, y_pred)
```

四、实验总结

本实验通过 PyTorch 的框架构建了神经网络，实现了波士顿房价预测的线性回归任务。回归是**监督学习**的一种任务，主要用于预测连续数值，它通过学习输入特征（X）与输出变量（Y）之间的关系，建立数学模型，从而对未知数据进行预测。通过该实验，使我对机器学习中“回归”的概念与应用有了更加深刻的认识。