Homework1 波士顿房价预测

一、实验背景与目标

本实验旨在构建一个综合性房价预测系统，通过多种机器学习模型对波士顿房价数据集进行分析和预测。实验具体目标包括：

1. 设计模块化的预测系统架构，支持多种特征工程和模型实现方法

2. 探索数据特征与房价的关系，找出关键影响因素

3. 比较不同类型模型在该任务上的适用性与性能

4. 总结小数据集建模的最佳实践经验

二、系统设计思路

2.1 系统整体架构

系统采用模块化设计，包含六个主要功能模块：数据加载与预处理模块：处理数据导入、划分和标准化；特征工程模块：实现相关性分析和PCA降维；基线模型模块：建立基准性能参考；模型训练模块：实现和训练多种预测模型；模型评估模块：统一的评估方法和指标计算；可视化模块：结果展示和模型比较。

2.2 数据预处理设计

数据预处理阶段采用以下设计理念：

1. 随机分割替代顺序分割：避免数据分布偏差，采用train\_test\_split函数确保随机性

2. 三重数据划分：训练集(64%)、验证集(16%)、测试集(20%)，各司其职

3. 特征标准化：对所有数值特征应用StandardScaler，保证各特征量纲一致性

4. 缺失值处理：虽然波士顿数据集无缺失值，但系统预留了缺失值处理机制

核心代码设计：

# 随机分割数据集（不再使用顺序分割）

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, y, test\_size=0.2, random\_state=42)

# 进一步分割训练集为训练和验证集

X\_train, X\_val, y\_train, y\_val = train\_test\_split(

X\_train, y\_train, test\_size=0.2, random\_state=42)

# 数据标准化

scaler = StandardScaler()

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

X\_val\_scaled = scaler.transform(X\_val)

X\_test\_scaled = scaler.transform(X\_test)

2.3 特征工程设计思路

特征工程模块设计中考虑了两条不同路径：

1. 专家知识路径：相关性分析。设计原理：利用统计知识筛选与目标强相关的特征。阈值选择：|r| > 0.5，平衡信息保留与维度减少。优势：模型可解释性强，计算成本低。

2. 自动特征路径：PCA降维。设计原理：保留数据主要方差，消除特征共线性。阈值选择：累积方差贡献率90%，通过变化曲线确定。优势：特征间去相关，降低模型复杂度

两种方法并行实现，以便比较不同特征工程策略的效果。

2.4 模型架构设计

系统设计包含五类不同复杂度和原理的模型：

1. 基线模型

设计思路：以目标变量均值作为预测值，建立性能下限

代码实现：自定义calculate\_baseline函数

2. 线性模型

设计思路：两个版本（相关性特征/PCA特征），验证线性假设

实现决策：使用sklearn的LinearRegression，简洁稳定

3. 集成树模型

设计思路：包含随机森林和梯度提升两种集成策略

参数设计：平衡树的数量(100)和深度(3)，避免过拟合

实现考虑：使用成熟的sklearn实现，专注于参数调优

4. 神经网络模型

设计思路：使用PyTorch自定义网络架构

架构优化：从(64→32→1)简化到(16→8→1)，增加dropout层

训练设计：加入早停机制和学习率调度器

5. 交叉验证模型

设计思路：通过5折交叉验证提升模型稳定性和泛化能力

实现考虑：独立训练多个模型实例，取平均性能

神经网络代码设计：

class SimpleNN(nn.Module):

def \_\_init\_\_(self, input\_size):

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

self.model = nn.Sequential(

nn.Linear(input\_size, 16),

nn.ReLU(),

nn.Dropout(0.3),

nn.Linear(16, 8),

nn.ReLU(),

nn.Dropout(0.3),

nn.Linear(8, 1) )

def forward(self, x):

return self.model(x)

2.5 训练过程设计

神经网络训练流程设计的核心考虑：

1. 批处理机制：

批量大小设为16（小于常规32），考虑到小数据集特点

使用DataLoader实现批处理，提高训练效率

2. 优化器选择：

采用Adam优化器，结合学习率0.0005和权重衰减0.01

权重衰减参数远高于默认值，针对小数据集强化正则化

3. 学习率调度：

使用ReduceLROnPlateau，监控验证损失自动调整学习率

配置patience=10，确保在验证损失不再下降时降低学习率

4. 早停机制：

设计patience=50，在验证损失不再改善时及时停止训练

保存最佳模型状态，避免过拟合阶段的模型劣化

训练流程设计代码：

# 设置训练参数

num\_epochs = 1000 # 设置较大的最大轮次

batch\_size = 16 # 小批量大小

patience = 50 # 早停耐心值

# 创建优化器和学习率调度器

optimizer = optim.Adam(model\_nn.parameters(), lr=0.0005, weight\_decay=0.01)

scheduler = ReduceLROnPlateau(optimizer, mode='min', factor=0.5, patience=10, verbose=True)

# 训练循环

best\_val\_loss = float('inf')

patience\_counter = 0

best\_model\_state = None

for epoch in range(num\_epochs):

# 训练阶段

model\_nn.train()

train\_epoch\_loss = 0

for X\_batch, y\_batch in train\_loader:

# 前向传播

y\_pred = model\_nn(X\_batch)

loss = criterion(y\_pred, y\_batch)

# 反向传播和优化

optimizer.zero\_grad()

loss.backward()

optimizer.step()

train\_epoch\_loss += loss.item()

# 验证阶段

model\_nn.eval()

with torch.no\_grad():

for X\_val\_batch, y\_val\_batch in val\_loader:

y\_val\_pred = model\_nn(X\_val\_batch)

val\_loss = criterion(y\_val\_pred, y\_val\_batch).item()

# 学习率调整

scheduler.step(val\_loss)

# 早停检查

if val\_loss < best\_val\_loss:

best\_val\_loss = val\_loss

patience\_counter = 0

best\_model\_state = model\_nn.state\_dict().copy()

else:

patience\_counter += 1

# 早停条件

if patience\_counter >= patience:

print(f'验证损失没有改善 {patience} 轮，提前停止训练')

break

# 加载最佳模型

if best\_model\_state:

model\_nn.load\_state\_dict(best\_model\_state)

三、系统实现与代码结构

3.1 代码组织结构

代码组织遵循功能模块化原则，按功能和数据流程划分为多个逻辑块：

try\_optimized.py

|-- 数据加载与预处理 (1)

|-- 相关性分析与特征选择 (2-3)

|-- PCA降维 (4)

|-- 基线模型实现 (5)

|-- 线性回归模型实现 (6)

|-- 神经网络模型定义 (7)

|-- 神经网络训练与早停 (8)

|-- 交叉验证神经网络 (9)

|-- 随机森林和梯度提升模型 (10-11)

|-- 最终测试评估 (12)

|-- 模型比较与可视化 (13)

3.2 关键模块功能说明

3.2.1 数据加载与预处理模块

3.2.2 特征工程模块

# 相关性分析

corr\_matrix = df.corr()

target\_corr = corr\_matrix[target\_col].sort\_values(ascending=False)

# 选择高相关性特征

threshold = 0.5

selected\_features = target\_corr[abs(target\_corr) > threshold].index.tolist()

selected\_features.remove(target\_col)

# 主成分分析

pca = PCA(n\_components=0.9) # 保留90%的方差

X\_train\_pca = pca.fit\_transform(X\_train\_scaled)

X\_test\_pca = pca.transform(X\_test\_scaled)

3.2.3 模型定义与训练模块

# 神经网络模型定义

class SimpleNN(nn.Module):

def \_\_init\_\_(self, input\_size):

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

self.model = nn.Sequential(

nn.Linear(input\_size, 16),

nn.ReLU(),

nn.Dropout(0.3),

nn.Linear(16, 8),

nn.ReLU(),

nn.Dropout(0.3),

nn.Linear(8, 1)

)

def forward(self, x):

return self.model(x)

# 线性模型训练

model = LinearRegression()

model.fit(X\_train\_selected, y\_train)

y\_pred\_selected = model.predict(X\_test\_selected)

# 随机森林模型训练

rf\_model = RandomForestRegressor(n\_estimators=100, random\_state=42)

rf\_model.fit(X\_train\_scaled, y\_train)

y\_pred\_rf = rf\_model.predict(X\_test\_scaled)

3.3 训练流程实现

神经网络训练实现了完整的训练、验证、早停和模型保存流程：

1. 数据加载器设置

2. 训练循环构建

3. 验证阶段设计

4. 早停条件检查

5. 最佳模型保存与加载

3.4 评估与可视化实现

评估模块统一实现了各模型的性能计算与比较：

# 模型评估

print("\n== 模型对比 ==")

print(f"基线模型 RMSE: {baseline\_rmse:.4f}")

print(f"线性回归 (相关性特征) RMSE: {selected\_rmse:.4f} | R²: {selected\_r2:.4f}")

print(f"线性回归 (PCA特征) RMSE: {pca\_rmse:.4f} | R²: {pca\_r2:.4f}")

print(f"随机森林 RMSE: {rf\_rmse:.4f} | R²: {rf\_r2:.4f}")

print(f"梯度提升树 RMSE: {gb\_rmse:.4f} | R²: {gb\_r2:.4f}")

print(f"神经网络 RMSE: {rmse\_nn:.4f} | R²: {r2\_nn:.4f}")

print(f"神经网络 (交叉验证) RMSE: {cv\_rmse:.4f} | R²: {cv\_r2:.4f}")

可视化模块实现了多种图表生成，用于模型比较与分析：

1. 训练损失曲线

2. 相关性热力图

3. PCA方差贡献曲线

4. 各模型预测结果散点图

5. 神经网络残差图

6. 随机森林特征重要性图表

四、实验结果

4.1 数据分析结果

4.1.1 相关性分析

与目标变量MEDV相关性最高的特征：

正相关：RM（0.695）- 每栋住宅的平均房间数

负相关：LSTAT（-0.738）- 低地位人口的百分比

负相关：PTRATIO（-0.508）- 城镇师生比例

4.1.2 PCA分析

保留90%方差需要7个主成分

第一主成分贡献率为42.68%

前三主成分累积贡献率达72.58%

4.2 模型性能对比

模型 | RMSE | R²

基线模型（均值预测） | 8.5635 | 0.0000

线性回归（相关性特征） | 5.2625 | 0.6224

线性回归（PCA特征） | 5.4583 | 0.5937

神经网络 | 6.2293 | 0.4709

神经网络（交叉验证） | 6.1243 | 0.5714

随机森林 | 3.4169 | 0.8408

梯度提升树 | 3.1925 | 0.8610

4.3 模型训练过程分析

神经网络训练过程显示：损失函数在前50轮迅速下降，验证损失在约70轮开始出现波动，学习率在第69轮自动降低为初始值的一半，训练在约第160轮由于早停机制终止。

五、分析与讨论

5.1 特征工程方法比较

相关性分析与PCA方法在线性模型上表现相近，说明高相关性特征包含了大部分有效信息，PCA虽降维50%以上，但保留了关键信息。

5.2 模型适用性分析

1. 树模型优势显著，梯度提升树和随机森林R²均超过0.84；能自动捕捉特征间的非线性关系；对小数据集表现出色。

2. 神经网络理想，优化后R²为0.57，但仍不如简单模型，主要受限于数据量不足（506条记录）。

3. 线性模型表现稳定，仅用三个特征达到0.62的R²；计算效率高，可解释性强，适合作为基准模型。

5.3 设计决策的影响分析

1. 数据划分方式：随机划分替代顺序划分，避免了分布偏移问题，测试证明这一改变显著提高了模型稳定性。

2. 神经网络架构简化：从深层(128→64→32→1)到浅层(16→8→1)，R²从-1.5555提升至0.4709，说明过度复杂导致过拟合

3. 正则化策略：权重衰减从1e-5提高到0.01（增强1000倍），Dropout从0.2提高到0.3，这些变化有效抑制了过拟合。

4. 早停机制：通过验证集监控，避免了训练过度，在最佳性能点保存模型，而非最后一轮。

六、结论与展望

6.1 主要结论

相关性分析虽简单但高效，三个高相关特征即可达到不错效果；PCA能有效减少特征数量但保留主要信息；随机森林特征重要性分析可作为特征选择的替代方法。系统设计经验：模块化设计使多模型对比与分析更加便捷，早停机制对小数据集神经网络训练至关重要，随机数据划分是保证模型泛化能力的基础。