**机器学习应用实践（实验五） —随机森林**

**工业智能2201班 刘天行 20225354**

1. **实验目的**

1、 熟悉随机森林算法的原理。

2、 掌握 Scikit-Learn 中随机森林 API 的使用， 理解算法中各参数的含义和作用。

3、掌握 Scikit-Learn 中提供的调参工具随机网格搜索和对半网格搜索的使用。

4、 具备使用 python 实现随机森林算法的编程能力。

1. **开发环境：Visual Studio Code + Jupyter notebook拓展 + python 3.12.2**

文本

描述已自动生成

1. **实验内容**
2. 题目一：采用 scikit-learn 中的 RandomForestRegressor 对加利福尼亚房价数据集进行预测。

【讨论一】比较随机森林和决策树在数据集上的表现。

【讨论二】随机森林中的 n\_estimator 超参数如何选择？

【讨论三】对于 RandomForestRegressor 模型， 自行选择超参数搜索的方法， 找到合适的超参数， 最终将超参数在如下的交叉验证集上进行建模，并计算 RMSE 评分。 介绍调参过程，并比较调参前后的效果。

1. 题目二： 编写随机森林算法，并对葡萄酒数据/加利福尼亚房价数据（只选择一种即可） 进行预测，并展示模型评分， 与 sklearn 自带的评估器建模结果进行对比。
2. **实验情况**

**题目一：**

**首先导入题目所需的数据集：**

from sklearn.datasets import fetch\_california\_housing

import time

from sklearn.model\_selection import GridSearchCV

from sklearn.ensemble import RandomForestClassifier

from sklearn.model\_selection import cross\_val\_score

from sklearn.model\_selection import validation\_curve

from sklearn.model\_selection import GridSearchCV

import numpy as np

# X\_california, y\_california = fetch\_california\_housing(return\_X\_y=True)

# 导入数据集

data = fetch\_california\_housing()

X\_california = data.data

y\_california = data.target

feature\_names = data.feature\_names

target\_name = "House Price"

# 输出数据集信息

print("数据集大小:", X\_california.shape)

print("特征数量:", len(feature\_names))

print("特征名称:", feature\_names)

print("标签名称:", target\_name)

print("标签分布情况:\n", data.target)

# 后续采用的是交叉验证，所以不需要划分训练集和测试集

**接下来，分别使用决策树和随机森林建立模型**

# 分别使用 DecisionTreeRegressor 和 RandomForestRegressor 建立分类模型（参数默认）

from sklearn.tree import DecisionTreeRegressor

from sklearn.ensemble import RandomForestRegressor

from sklearn.model\_selection import cross\_validate

# 建立DecisionTreeRegressor模型

decision\_tree\_reg = DecisionTreeRegressor()

# 以根均方误差 RMSE 为评估指标： ‘neg\_root\_mean\_squared\_error’

dt\_scores = cross\_validate(decision\_tree\_reg, X\_california, y\_california,

                            cv=10, scoring='neg\_root\_mean\_squared\_error', return\_train\_score=True,n\_jobs=-1)

dt\_train\_rmse = np.sqrt(-dt\_scores['train\_score'])

dt\_test\_rmse = np.sqrt(-dt\_scores['test\_score'])

print("DecisionTreeRegressor模型的训练集RMSE：", dt\_train\_rmse)

print("DecisionTreeRegressor模型的测试集RMSE：", dt\_test\_rmse)

print("DecisionTreeRegressor模型的拟合分别耗时：", dt\_scores['fit\_time'])

# 建立RandomForestRegressor模型

random\_forest\_reg = RandomForestRegressor()

# 以根均方误差 RMSE 为评估指标

rf\_scores = cross\_validate(random\_forest\_reg, X\_california, y\_california,

                            cv=10, scoring='neg\_root\_mean\_squared\_error', return\_train\_score=True,n\_jobs=-1)

rf\_train\_rmse = np.sqrt(-rf\_scores['train\_score'])

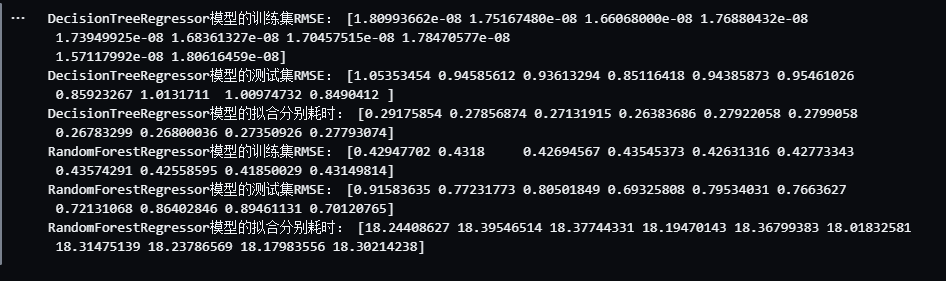
rf\_test\_rmse = np.sqrt(-rf\_scores['test\_score'])

print("RandomForestRegressor模型的训练集RMSE：", rf\_train\_rmse)

print("RandomForestRegressor模型的测试集RMSE：", rf\_test\_rmse)

print("RandomForestRegressor模型的拟合分别耗时：", rf\_scores['fit\_time'])

**运行代码，可以得到如下的结果：**



**以交叉验证的根均方误差 RMSE 为评估指标，根据运行结果我们可以发现在训练集上决策树的RMSE普遍大于随机森林的RMSE，但是随机森林的拟合耗时远多于决策树**

**讨论一：**

**为了更加直观地表示上述结果，我们将上述的结果使用折线图的形式可视化**

import matplotlib.pyplot as plt

# 提取决策树和随机森林的评分

dt\_test\_scores = dt\_scores['test\_score']

rf\_test\_scores = rf\_scores['test\_score']

# 绘制折线图

plt.plot( dt\_test\_scores, label='Decision Tree')

plt.plot( rf\_test\_scores, label='Random Forest')

# 添加图例、标签和标题

plt.legend()

plt.xlabel('Cross-validation Fold')

plt.ylabel('R^2 Score')

plt.title('Performance Comparison - Decision Tree vs Random Forest')

# 显示图形

plt.show()

# 如下图，可以看出随机森林的性能要优于决策树，且随着交叉验证的进行，随机森林的性能变化不大，而决策树的性能则有所波动

# 随机森林模型相对于单个决策树模型，在数据集上表现更好，其在训练集上具有更好的拟合能力，

# 并且在测试集上具有更低的预测误差，虽然牺牲了一些拟合时间。

**得到这样的折线图**

****

**我们在前面的总结基础上可以进一步发现,随机森林的性能要优于决策树，且随着交叉验证的进行，随机森林的性能变化不大，而决策树的性能则有所波动,随机森林模型相对于单个决策树模型，在数据集上表现更好，其在训练集上具有更好的拟合能力，并且在测试集上具有更低的预测误差，虽然牺牲了一些拟合时间。**

**讨论二：**

**为了更直观地看出随机森林中n\_estimator 超参数对模型性能的影响，我采用循环尝试不同的n\_estimator值分别训练随机森林模型并进行交叉验证，对比模型的RMSE来作为参数寻优的依据**

# n\_estimators 随机森林中决策树的数量

import numpy as np

import matplotlib.pyplot as plt

from sklearn.ensemble import RandomForestRegressor

from sklearn.model\_selection import cross\_val\_score

from tqdm import tqdm

# 定义 n\_estimators 的取值范围

n\_estimators\_range = range(1, 101)

# 初始化空列表，用于存储不同 n\_estimators 对应的交叉验证结果

mean\_scores = []

# 循环尝试不同的 n\_estimators 值

#for n\_estimators in n\_estimators\_range:

for n\_estimators in tqdm(n\_estimators\_range, desc="Training Progress"):

#增加训练进度条

    random\_forest\_reg = RandomForestRegressor(n\_estimators=n\_estimators, n\_jobs=-1)

    scores = cross\_val\_score(random\_forest\_reg, X\_california, y\_california, cv=10,

                         scoring='neg\_root\_mean\_squared\_error', n\_jobs=-1)

    # 创建随机森林回归模型

    ##random\_forest\_reg = RandomForestRegressor(n\_estimators=n\_estimators)

    # 执行交叉验证并计算评估指标（这里使用负的均方根误差）

    ##scores = cross\_val\_score(random\_forest\_reg, X\_california, y\_california, cv=10,

    #                          scoring='neg\_root\_mean\_squared\_error')

    # 计算均值并将结果添加到列表中

    mean\_scores.append(np.mean(scores))

# 绘制学习曲线

plt.plot(n\_estimators\_range, mean\_scores)

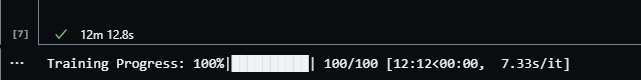
plt.xlabel('n\_estimators')

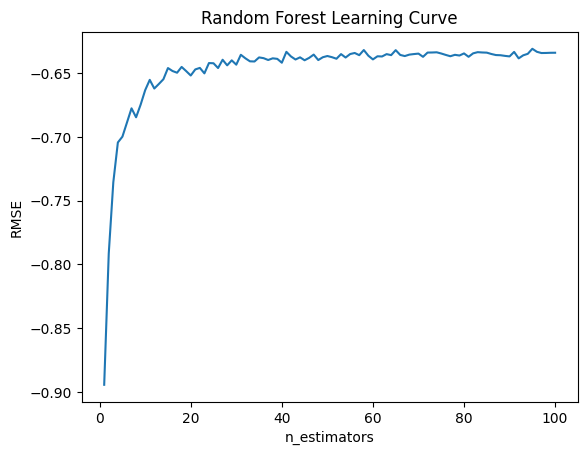
plt.ylabel('RMSE')

plt.title('Random Forest Learning Curve')

plt.show()

**在第一次尝试中，并未加入训练进度条，并且使用CPU单核训练，训练中存在效率低下，不知道训练进度的问题，因此我修改了代码，加入进度条来可视化训练进度，同时加入n\_jobs=-1的参数来调度全部的CPU核心参与训练，训练效率大幅提升，运行结果如下：**



****

import numpy as np

import matplotlib.pyplot as plt

from sklearn.ensemble import RandomForestRegressor

from sklearn.model\_selection import cross\_val\_score

from tqdm import tqdm

# 定义 n\_estimators 的取值范围

n\_estimators\_range = range(1,500, 5)

# 初始化空列表，用于存储不同 n\_estimators 对应的交叉验证结果

mean\_scores = []

# 循环尝试不同的 n\_estimators 值

#for n\_estimators in n\_estimators\_range:

for n\_estimators in tqdm(n\_estimators\_range, desc="Training Progress"):

    random\_forest\_reg = RandomForestRegressor(n\_estimators=n\_estimators, n\_jobs=-1)

    scores = cross\_val\_score(random\_forest\_reg, X\_california, y\_california, cv=10,

                         scoring='neg\_root\_mean\_squared\_error', n\_jobs=-1)

    # 创建随机森林回归模型

    #random\_forest\_reg = RandomForestRegressor(n\_estimators=n\_estimators)

    # 执行交叉验证并计算评估指标（这里使用负的均方根误差）

    #scores = cross\_val\_score(random\_forest\_reg, X\_california, y\_california, cv=10, scoring='neg\_root\_mean\_squared\_error')

    # 计算均值并将结果添加到列表中

    mean\_scores.append(np.mean(scores))

# 绘制学习曲线

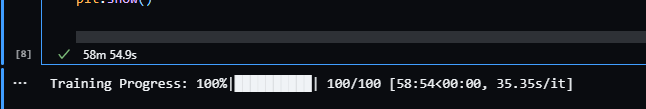
plt.plot(n\_estimators\_range, mean\_scores)

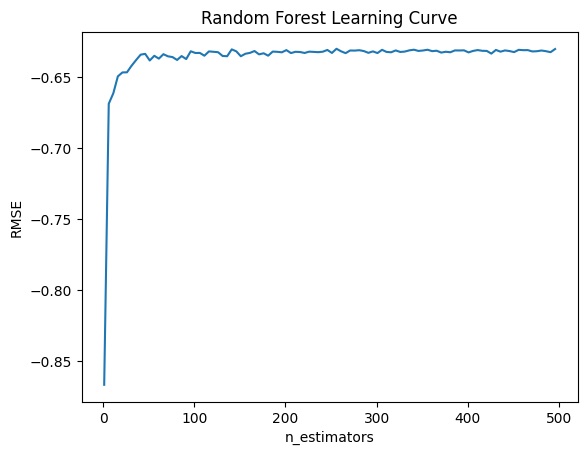
plt.xlabel('n\_estimators')

plt.ylabel('RMSE')

plt.title('Random Forest Learning Curve')

plt.show()



****

**绘制出0～100和0～500的决策树数量值和RMSE值的相关关系，大致确探索范围与变化趋势，导出结果存为csv文件**

import csv

# 将结果导出为 CSV 文件

output\_file = 'mean\_scores.csv'

with open(output\_file, 'w', newline='') as file:

    writer = csv.writer(file)

    writer.writerow(['n\_estimators', 'RMSE'])

    writer.writerows(zip(n\_estimators\_range, mean\_scores))

**选取200-400的区间使用RandomizedSearchCV进行寻优**

from sklearn.model\_selection import RandomizedSearchCV

# 定义 n\_estimators 的范围

n\_estimators\_range = list(range(200, 400))

# 创建随机森林回归模型

random\_forest\_reg = RandomForestRegressor()

# 定义超参数空间

param\_space = {

    'n\_estimators': n\_estimators\_range,

    # 添加其他超参数及其取值范围

}

# 执行随机搜索

random\_search = RandomizedSearchCV(random\_forest\_reg, param\_space, n\_iter=10, cv=10,

                                   scoring='neg\_root\_mean\_squared\_error', random\_state=520,n\_jobs=-1)

# 在数据集上进行拟合和超参数搜索

random\_search.fit(X\_california, y\_california)

# 输出最佳的超参数和评分

best\_params = random\_search.best\_params\_

best\_score = np.sqrt(-random\_search.best\_score\_)

print("最佳超参数:", best\_params)

print("最佳评分(RMSE):", -random\_search.best\_score\_)

# RandomizedSearchCV 的计算效率通常比 GridSearchCV 高，尤其是当超参数空间较大时

# GridSearchCV 对所有可能的参数组合进行 exhaustive 搜索，

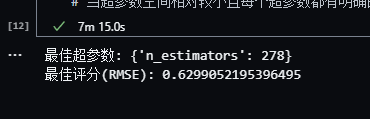
# 而 RandomizedSearchCV 则在给定的超参数空间中随机采样一组参数组合进行搜索

# 这对于一些超参数对模型性能有较大影响的情况下特别有用

# 然而，GridSearchCV 也有其优势，它遍历整个参数空间，不会错过任何可能的组合

# 当超参数空间相对较小且每个超参数都有明确的取值范围时，GridSearchCV 可以确保找到最优解。

**可以得到最佳超参数和对应的最佳评分**



**讨论三：**

from sklearn.model\_selection import RandomizedSearchCV

from scipy.stats import randint

# 定义超参数搜索范围

param\_dist = {

    'n\_estimators': randint(90, 200),  # 决策树的数量,randint会生成50-200之间的随机整数

    'max\_depth': [None, 1, 5, 10],  # 每棵树的最大深度

}

# 建立RandomForestRegressor模型

random\_forest\_reg = RandomForestRegressor()

# 使用RandomizedSearchCV进行超参数搜索

random\_search = RandomizedSearchCV(random\_forest\_reg, param\_distributions=param\_dist,

                                   scoring='neg\_root\_mean\_squared\_error', cv=10, n\_iter=10,n\_jobs=-1)

random\_search.fit(X\_california, y\_california)

# 输出最佳超参数组合和对应的RMSE评分

print("最佳超参数组合：", random\_search.best\_params\_)

print("最佳评分(RMSE):", -random\_search.best\_score\_)

# print("最佳RMSE评分：", np.sqrt(-random\_search.best\_score\_))

**思路与上面的题目基本一致，最终得到结果如下：**

文本

描述已自动生成

**题目二：**

**首先导入数据集**

from sklearn.datasets import fetch\_california\_housing

import time

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

from sklearn.model\_selection import GridSearchCV

from sklearn.ensemble import RandomForestClassifier

from sklearn.model\_selection import cross\_val\_score

from sklearn.model\_selection import validation\_curve

from sklearn.model\_selection import GridSearchCV

import numpy as np

from sklearn import tree

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

import matplotlib.pyplot as plt

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

from sklearn.metrics import mean\_squared\_error,r2\_score

# X\_california, y\_california = fetch\_california\_housing(return\_X\_y=True)

# 导入数据集

data = fetch\_california\_housing()

X\_california = data.data

y\_california = data.target

feature\_names = data.feature\_names

target\_name = "House Price"

# 输出数据集信息

print("数据集大小:", X\_california.shape)

print("特征数量:", len(feature\_names))

print("特征名称:", feature\_names)

print("标签名称:", target\_name)

print("标签分布情况:\n", data.target)

文本

描述已自动生成

**之后，编写随机森林算法，使用上一次实验中编写的CART\_decisio\_tree决策树来实现单棵决策树**

class RandomForest():

    # 构造函数，树的数量，树的类型，树的特征数，树的最大深度，树的特征名称

    def \_\_init\_\_(self, n\_samples=506, n\_chosen\_features=8, n\_features=13,

                  n\_tree=10, tree='mse', criterion='mse',max\_depth=None, featurenames=None) -> None:

        self.n\_tree = n\_tree

        self.tree = tree

        self.criterion = criterion

        self.max\_depth = max\_depth

        self.n\_features = n\_features

        self.n\_chosen\_features = n\_chosen\_features

        self.featurenames = featurenames

        self.forest = []

        self.n\_samples = n\_samples

        self.train\_idx = np.random.choice(n\_samples,n\_samples,replace=True)

        self.val\_idx = np.setdiff1d(np.arange(0,n\_samples), np.unique(self.train\_idx))

        self.feature\_idx = np.zeros((self.n\_tree,self.n\_chosen\_features)).astype(np.int)

        for i in range(self.n\_tree):

            self.feature\_idx[i,:] = np.random.choice(n\_features,self.n\_chosen\_features, replace=False)

        print(self.feature\_idx)

        # print(self.val\_idx)

    def grow(self, X, y):

    # 训练函数，输入训练集，输出训练好的随机森林

        for i in range(self.n\_tree):

            print('\*'\*20,i,'th tree','\*'\*20)

            # 调用决策树类，训练决策树

            regressionTree = CART\_decision\_tree(tree=self.tree, criterion=self.criterion,\

                                                max\_depth=self.max\_depth,featurenames=self.featurenames)

            regressionTree.fit(X[self.train\_idx][:,self.feature\_idx[i]], y[self.train\_idx])

            self.forest.append(regressionTree)

    def predict(self,X):

        pre = np.zeros([self.n\_tree,X.shape[0]])

        for i in range(self.n\_tree):

            pre[i,:] = self.forest[i].predict(X[:,self.feature\_idx[i]])

        # print(pre)

        return pre.mean(axis=0)

    def predict\_val(self,X):

        pre = np.zeros([self.n\_tree,len(self.val\_idx)])

        for i in range(self.n\_tree):

            pre[i,:] = self.forest[i].predict(X[self.val\_idx,:][:,self.feature\_idx[i]])

        return pre.mean(axis=0)

    # 评分函数，输入测试集，输出测试集的评分

    def score\_val(self, X, y, crioterion='mse'):

        pre\_val = self.predict\_val(X)

        if crioterion=='mse':

            return mean\_squared\_error(pre\_val, y[self.val\_idx])

        else:

            return r2\_score(pre\_val, y[self.val\_idx])

    # 评分函数，输入测试集，输出测试集的评分

    def score(self, pre, gt, crioterion='mse'):

        if crioterion=='mse':

            return mean\_squared\_error(pre, gt)

        else:

            return r2\_score(pre, gt)

# 2、数据集划分

# 这里需要显示划分训练集和测试集，因为这个模型的手写实现没有内置的交叉验证

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_california, y\_california, test\_size=0.2, random\_state=520)

# 3、创建回归树实例

regressionTree = CART\_decision\_tree(tree='mse', criterion='mse',max\_depth=3,featurenames=feature\_names)

# 4、训练模型（调用fit)

regressionTree.fit(X\_train, y\_train)

# regressionTree.fit(X\_test, y\_test)

# 5、输出树模型

regressionTree.print\_tree()

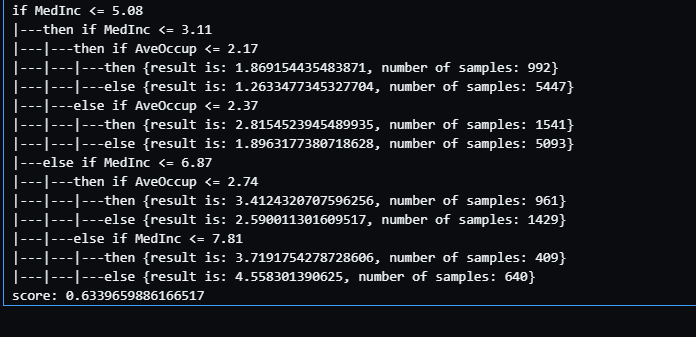
# 6、预测，评分

pre\_test = regressionTree.predict(X\_test)

score = regressionTree.score(pre\_test, y\_test)

print("score:", score)

**得到下面的结果：**



**关键代码片段：**

**引导抽样，在训练每棵树时，从训练数据中有放回地抽取样本：**

self.train\_idx = np.random.choice(n\_samples, n\_samples, replace=True)

self.val\_idx = np.setdiff1d(np.arange(0, n\_samples), np.unique(self.train\_idx))

**随机选择特征，在每棵树的节点上，从所有特征中随机选择一部分特征用于分裂：**

self.feature\_idx = np.zeros((self.n\_tree, self.n\_chosen\_features)).astype(np.int)

for i in range(self.n\_tree):

self.feature\_idx[i,:] = np.random.choice(n\_features, self.n\_chosen\_features, replace=False)

**使用CART算法生成每颗决策树：**

regressionTree = CART\_decision\_tree(tree=self.tree, criterion=self.criterion, max\_depth=self.max\_depth, featurenames=self.featurenames)

regressionTree.fit(X[self.train\_idx][:,self.feature\_idx[i]], y[self.train\_idx])

self.forest.append(regressionTree)

**预测，通过每棵树预测结果的平均值得到最终的预测结果：**

def predict(self, X):

pre = np.zeros([self.n\_tree, X.shape[0]])

for i in range(self.n\_tree):

pre[i,:] = self.forest[i].predict(X[:, self.feature\_idx[i]])

return pre.mean(axis=0)

**模型评估，使用MSE和R2来分析和评估模型性能：**

def score\_val(self, X, y, crioterion='mse'):

pre\_val = self.predict\_val(X)

if crioterion=='mse':

return mean\_squared\_error(pre\_val, y[self.val\_idx])

else:

return r2\_score(pre\_val, y[self.val\_idx])

def score(self, pre, gt, crioterion='mse'):

if crioterion=='mse':

return mean\_squared\_error(pre, gt)

else:

return r2\_score(pre, gt)

**与sklearn自带的随机森林进行对比**

from sklearn.ensemble import RandomForestRegressor

from sklearn.model\_selection import cross\_validate

# 建立RandomForestRegressor模型

random\_forest\_reg = RandomForestRegressor()

# 以根均方误差 RMSE 为评估指标

rf\_scores = cross\_validate(random\_forest\_reg, X\_california, y\_california,

                            cv=10, scoring='neg\_root\_mean\_squared\_error', return\_train\_score=True)

rf\_train\_rmse = np.sqrt(-rf\_scores['train\_score'])

rf\_test\_rmse = np.sqrt(-rf\_scores['test\_score'])

print("RandomForestRegressor模型的训练集RMSE：", rf\_train\_rmse)

print("RandomForestRegressor模型的测试集RMSE：", rf\_test\_rmse)

print("RandomForestRegressor模型的拟合分别耗时：", rf\_scores['fit\_time'])

文本

描述已自动生成

**可见，手动编写的随机森林在一定程度上拟合效果超越了sklearn内置的随机森林。**