《机器学习基础》实验报告

年级、专业、班级	2021 级计算机科学与技术 4 班	姓名	胡鑫
实验题目	决策树算法实践		
实验时间	2023 年 5 月 11 日	实验地点	DS3402
实验成绩		实验性质	☐验证性 ☐设计性 ☐综合性

教师评价：

☐算法/实验过程正确； ☐源程序/实验内容提交 ☐程序结构/实验步骤合理；

☐实验结果正确； ☐语法、语义正确； ☐报告规范；

其他：

评价教师签名：

一、实验目的

掌握决策树回归算法原理。

二、实验项目内容

1.理解并**描述**决策树分类、回归算法原理。

2.**编程**实践，将决策树分类、回归算法分别应用于合适的数据集(如鸢尾花、波士顿房价预测、UCI 数据集、Kaggle 数据集)，要求算法至少用于两个数据集(分类 2 个，回归 2 个)。

三、实验过程或算法（源程序）

1. 决策树算法将数据集划分为多个小的决策单元，根据每个决策单元的特征判断其属于哪一个类别或数值范围。

（1）在分类问题中，决策树算法通过比较输入特征的不同值，将数据集划分为不同的分类，直到所有数据都被分成相同类别为止。决策树的分类过程可以用一个树状结构来表示，每个决策单元对应一个节点，每个节点包含多个分支，每个分支代表一个特征值。在分类阶段，根据输入特征的取值，从根节点开始经过多个分支，最终到达某个叶子节点，这个叶子节点对应的类别即为分类结果。决策树分类算法基于信息熵和信息增益的概念来确定最佳的划分特征，其中信息熵表示数据集的不确定度，信息增益表示使用某个特征进行划分后，数据集的不确定度减少了多少。假定当前样本集合 D 中第 k 类样本所占的比例为 p_k(k=1,2,…|y|),则 D 的信息熵定义为 $\mathrm{Ent}(D) = -\sum_{k=1}^{|\mathcal{Y}|} p_k \log_2 p_k$.,假定离散属性 a 有 V 个可能的取值$\{a^1,a^2,\ldots,a^V\}$，若使用 a 来对样本集 D 进行划分，则会产生 V 个分支节点，其中第 v 个分支节点包含了 D 中所有在属性 a 上取值为 a^v 的样本，记为 D^v，则可根据信息熵公式计算出 D 的信息熵，于是可计算出用

属性 a 对样本集 D 进行划分所获得的信息增益 $$\mathrm{Gain}(D,a)=\mathrm{Ent}(D)-\sum_{v=1}^{V}\frac{|D^v|}{|D|}\mathrm{Ent}(D^v)\ .$$

（2）在回归问题中，决策树算法通过比较输入特征的不同值，将数据集划分为不同数值范围，直到所有数据都落入同一个范围内。决策树回归算法基于基尼系数和分裂变化量的概念来确定最佳的划分特征，其中基尼系数表示数据集的不确定性，分裂变化量表示使用某个特征进行划分后，数据集的不确定性发生了多大的变化。CART 决策树使用基尼指数来选择划分属性，

$$\begin{aligned}\mathrm{Gini}(D)&=\sum_{k=1}^{|\mathcal{Y}|}\sum_{k'\neq k}p_k p_{k'}\\&=1-\sum_{k=1}^{|\mathcal{Y}|}p_k^2\ .\end{aligned}$$

，属性 a 的基尼指数定义为 $$\mathrm{Gini\_index}(D,a)=\sum_{v=1}^{V}\frac{|D^v|}{|D|}\mathrm{Gini}(D^v)\ .$$,在候选属性集合 A 中，选择那个使得划分后基尼指数最小的属性作为最优划分属性，即 $a_*=\underset{a\in A}{\arg\min}\ \mathrm{Gini\_index}(D,a).$

决策树算法具有可解释性好、易于理解和实现等优点，但在面对高维度和复杂数据时容易出现过拟合。为了解决过拟合问题，可以通过剪枝、随机森林等方式进行改进。

2.（1）鸢尾花分类

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
# 定义节点类
class Node:
    def __init__(self, y_values=None, feature_ind=None, feature_val=None, left=None, right=None, is_leaf=True, class_label=None):
        self.y_values = y_values  # 节点包含的样本标签
        self.feature_ind = feature_ind  # 节点划分所用特征的索引
        self.feature_val = feature_val  # 节点划分所用特征的取值
        self.left = left  # 左子树
        self.right = right  # 右子树
        self.is_leaf = is_leaf  # 是否为叶节点
        self.class_label = class_label  # 叶节点的类别标签

# 定义决策树分类器类
class DecisionTreeClassifier:
    def __init__(self, max_depth):
        self.max_depth = max_depth  # 树的最大深度

    # 训练决策树模型
    def fit(self, X, y):
        self.n_classes = len(np.unique(y))  # 类别数
        self.n_features = X.shape[1]  # 特征数
```

```
        self.tree_ = self.build_tree(X, y)    # 构建决策树

    # 决策树预测函数
    def predict(self, X):
        n_samples = X.shape[0]
        y_pred = np.empty((n_samples,), dtype=int)    # 预测结果

        for i, x in enumerate(X):
            node = self.tree_
            while not node.is_leaf:
                if x[node.feature_ind] <= node.feature_val:
                    node = node.left
                else:
                    node = node.right
            y_pred[i] = node.class_label

        return y_pred

    # 计算基尼指数
    def get_gini(self, y):
        # 计算数据集的基尼指数
        _, counts = np.unique(y, return_counts=True)
        p = counts / len(y)
        gini = 1 - np.sum(p ** 2)

        return gini

    # 计算加权基尼指数
    def get_weighted_gini(self, X_left, X_right, y_left, y_right):
        # 计算划分后的样本基尼指数，并加权平均
        p_left = len(y_left) / (len(y_left) + len(y_right))
        p_right = len(y_right) / (len(y_left) + len(y_right))
        g_left = self.get_gini(y_left)
        g_right = self.get_gini(y_right)
        wgini = p_left * g_left + p_right * g_right

        return wgini

    # 对数据进行划分并构建决策树
    def build_tree(self, X, y, depth=0):
        n_samples, n_features = X.shape

        # 判断是否达到叶子节点
        if (depth == self.max_depth) or (self.n_classes == 1):
```

```
            label = np.argmax(np.bincount(y))  # 返回频数最大的类别
            return Node(y_values=y, is_leaf=True, class_label=label)

        # 在所有特征中找到最佳划分特征和划分值
        best_feature = None
        best_value = None
        best_score = np.inf
        for feature in range(n_features):
            X_feature = X[:, feature]
            vals = np.unique(X_feature)

            # 生成划分方案
            for split_val in vals:
                # 根据划分方案分离出左右两侧数据
                y_left = y[X_feature <= split_val]
                y_right = y[X_feature > split_val]
                if (len(y_left) == 0) or (len(y_right) == 0):
                    continue

                # 计算划分方案的加权基尼指数
                gini_score = self.get_weighted_gini(X[X_feature <= split_val], X[X_feature > split_val], y_left, y_right)

                # 根据加权基尼指数找到最佳划分规则
                if gini_score < best_score:
                    best_feature = feature
                    best_value = split_val
                    best_score = gini_score

        # 判断是否达到叶子节点
        if best_feature is None:
            label = np.argmax(np.bincount(y))  # 返回频数最大的类别
            return Node(y_values=y, is_leaf=True, class_label=label)

        # 根据最佳划分规则分离出左右两侧数据和标签
        X_left = X[X[:, best_feature] <= best_value]
        y_left = y[X[:, best_feature] <= best_value]
        X_right = X[X[:, best_feature] > best_value]
        y_right = y[X[:, best_feature] > best_value]

        # 生成左右子树
        node = Node(y_values=y, feature_ind=best_feature, feature_val=best_value, is_leaf=False)
        node.left = self.build_tree(X_left, y_left, depth+1)
        node.right = self.build_tree(X_right, y_right, depth+1)
```

```
            return node


# 从 csv 文件中读取鸢尾花数据集
data = pd.read_csv('Lab3\iris.csv')

# 将特征和标签拆分开
X = data.iloc[:, :-1].values
y = data.iloc[:, -1].values

# 将数据集分成训练集和测试集
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# 使用 CART 算法进行训练和预测
tree = DecisionTreeClassifier(max_depth=3)
tree.fit(X_train, y_train)
y_pred = tree.predict(X_test)

# 计算模型准确率
acc = np.sum(y_pred == y_test) / len(y_test)
print("Accuracy:", acc)
```

（2）红酒分类

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split

# 定义节点类
class Node:
    def __init__(self, y_values=None, feature_ind=None, feature_val=None, left=None, right=None,
is_leaf=True, class_label=None):
        self.y_values = y_values  # 节点包含的样本标签
        self.feature_ind = feature_ind  # 节点划分所用特征的索引
        self.feature_val = feature_val  # 节点划分所用特征的取值
        self.left = left  # 左子树
        self.right = right  # 右子树
        self.is_leaf = is_leaf  # 是否为叶节点
        self.class_label = class_label  # 叶节点的类别标签

# 定义决策树分类器类
class DecisionTreeClassifier:
    def __init__(self, max_depth):
        self.max_depth = max_depth  # 树的最大深度
```

```
# 训练决策树模型
def fit(self, X, y):
    self.n_classes = len(np.unique(y))  # 类别数
    self.n_features = X.shape[1]  # 特征数
    self.tree_ = self.build_tree(X, y)  # 构建决策树

# 决策树预测函数
def predict(self, X):
    n_samples = X.shape[0]
    y_pred = np.empty((n_samples,), dtype=int)  # 预测结果

    for i, x in enumerate(X):
        node = self.tree_
        while not node.is_leaf:
            if x[node.feature_ind] <= node.feature_val:
                node = node.left
            else:
                node = node.right
        y_pred[i] = node.class_label

    return y_pred

# 计算基尼指数
def get_gini(self, y):
    # 计算数据集的基尼指数
    _, counts = np.unique(y, return_counts=True)
    p = counts / len(y)
    gini = 1 - np.sum(p ** 2)

    return gini

# 计算加权基尼指数
def get_weighted_gini(self, X_left, X_right, y_left, y_right):
    # 计算划分后的样本基尼指数，并加权平均
    p_left = len(y_left) / (len(y_left) + len(y_right))
    p_right = len(y_right) / (len(y_left) + len(y_right))
    g_left = self.get_gini(y_left)
    g_right = self.get_gini(y_right)
    wgini = p_left * g_left + p_right * g_right

    return wgini

# 对数据进行划分并构建决策树
```

```
    def build_tree(self, X, y, depth=0):
        n_samples, n_features = X.shape

        # 判断是否达到叶子节点
        if (depth == self.max_depth) or (self.n_classes == 1):
            label = np.argmax(np.bincount(y))  # 返回频数最大的类别
            return Node(y_values=y, is_leaf=True, class_label=label)

        # 在所有特征中找到最佳划分特征和划分值
        best_feature = None
        best_value = None
        best_score = np.inf
        for feature in range(n_features):
            X_feature = X[:, feature]
            vals = np.unique(X_feature)

            # 生成划分方案
            for split_val in vals:
                # 根据划分方案分离出左右两侧数据
                y_left = y[X_feature <= split_val]
                y_right = y[X_feature > split_val]
                if (len(y_left) == 0) or (len(y_right) == 0):
                    continue

                # 计算划分方案的加权基尼指数
                gini_score = self.get_weighted_gini(X[X_feature <= split_val], X[X_feature > split_val], y_left, y_right)

                # 根据加权基尼指数找到最佳划分规则
                if gini_score < best_score:
                    best_feature = feature
                    best_value = split_val
                    best_score = gini_score

        # 判断是否达到叶子节点
        if best_feature is None:
            label = np.argmax(np.bincount(y))  # 返回频数最大的类别
            return Node(y_values=y, is_leaf=True, class_label=label)

        # 根据最佳划分规则分离出左右两侧数据和标签
        X_left = X[X[:, best_feature] <= best_value]
        y_left = y[X[:, best_feature] <= best_value]
        X_right = X[X[:, best_feature] > best_value]
        y_right = y[X[:, best_feature] > best_value]
```

```
            # 生成左右子树
            node = Node(y_values=y, feature_ind=best_feature, feature_val=best_value, is_leaf=False)
            node.left = self.build_tree(X_left, y_left, depth+1)
            node.right = self.build_tree(X_right, y_right, depth+1)

            return node


# 从 csv 文件中读取红酒数据集
data = pd.read_csv('Lab3\wine.csv')

# 将特征和标签拆分开
X = data.iloc[:, :-1].values
y = data.iloc[:, -1].values

# 将数据集分成训练集和测试集
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.125, random_state=42)

# 使用 CART 算法进行训练和预测
tree = DecisionTreeClassifier(max_depth=13)
tree.fit(X_train, y_train)
y_pred = tree.predict(X_test)

# 计算模型准确率
acc = np.sum(y_pred == y_test) / len(y_test)
print("Accuracy:", acc)
```

（3）波士顿房价预测

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt


# ---------- CART 回归树算法 ----------

# 定义节点类
class Node:
    def __init__(self, split_feature, split_value, left_child, right_child, prediction):
        self.split_feature = split_feature
        self.split_value = split_value
        self.left_child = left_child
        self.right_child = right_child
```

```
        self.prediction = prediction

# 计算均方误差
def mse(y):
    return np.mean((y - np.mean(y))**2)

# 递归构建 CART 树
def build_CART_tree(X, y, min_samples_leaf=2, depth=0):
    # 如果样本数达到叶子节点要求的最小值，则直接返回叶子节点的平均值
    if len(y) <= min_samples_leaf:
        return Node(None, None, None, None, np.mean(y))

    # 如果树的深度达到最大值，则直接返回叶子节点的平均值
    if depth >= max_depth:
        return Node(None, None, None, None, np.mean(y))

    # 寻找最优的分裂特征和分裂阈值
    best_feature, best_value, best_mse = None, None, np.inf
    for i in range(X.shape[1]):
        for value in np.unique(X[:,i]):
            left_indices = X[:,i] <= value
            right_indices = X[:,i] > value
            left_mse = mse(y[left_indices])
            right_mse = mse(y[right_indices])
            mse_sum = left_mse*(sum(left_indices)/len(X)) +
right_mse*(sum(right_indices)/len(X))
            if mse_sum < best_mse:
                best_feature, best_value, best_mse = i, value, mse_sum

    # 如果没有找到满足分裂条件的节点，则直接返回叶子节点的平均值
    if best_feature is None:
        return Node(None, None, None, None, np.mean(y))

    # 根据最优的分裂特征和分裂阈值将数据集分为左右两个子树
    left_indices = X[:,best_feature] <= best_value
    right_indices = X[:,best_feature] > best_value
    left_X, left_y = X[left_indices], y[left_indices]
    right_X, right_y = X[right_indices], y[right_indices]

    # 递归构建左右两个子树
    left_child = build_CART_tree(left_X, left_y, min_samples_leaf, depth+1)
    right_child = build_CART_tree(right_X, right_y, min_samples_leaf, depth+1)

    # 返回当前节点
```

```
        return Node(best_feature, best_value, left_child, right_child, np.mean(y))

# 对新数据进行预测
def predict(node, x):
        if node.split_feature is None:
                return node.prediction
        elif x[node.split_feature] <= node.split_value:
                return predict(node.left_child, x)
        else:
                return predict(node.right_child, x)

# ------------ 执行程序 -------------

# 构建 CART 树
max_depth = 10
min_samples_leaf = 7
# 从 CSV 文件读取数据
df = pd.read_csv('Lab3/housing.csv')
df = df.dropna()  # 删除空值所在的行
X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.13, random_state=42)
tree_root = build_CART_tree(X_train, y_train, min_samples_leaf)

# 在测试集上进行预测，并计算均方误差
y_pred = np.zeros(len(y_test))
for i in range(len(y_test)):
        y_pred[i] = predict(tree_root, X_test[i])
mse_test = mse(y_test - y_pred)
print("均方误差: {}".format(mse_test))

# 使用模型进行预测
y_pred_train = np.zeros(len(y_train))
y_pred_test = np.zeros(len(y_test))
for i in range(len(y_train)):
        y_pred_train[i] = predict(tree_root, X_train[i])
for i in range(len(y_test)):
        y_pred_test[i] = predict(tree_root, X_test[i])

# 绘制训练集和测试集的真实值和预测值对比图
plt.figure(figsize=(8, 8))
plt.scatter(y_train, y_pred_train, label='Training set')
plt.scatter(y_test, y_pred_test, label='Testing set')
plt.plot([np.min(y), np.max(y)], [np.min(y), np.max(y)], 'k--', lw=2)
```

```
plt.xlabel("True Values")
plt.ylabel("Predictions")
plt.title("True vs Predicted values")
plt.legend(loc="upper left")
plt.show()
```

（4）特斯拉股票预测

```
import csv
import numpy as np
import matplotlib.pyplot as plt

# 读入数据
dates = []
data = []
with open('Lab3\Tesla.csv') as f:
    reader = csv.reader(f)
    # 解析表头
    header = next(reader)
    # 解析数据
    for row in reader:
        dates.append(row[0])
        data.append(row[1:])

# 转换数据格式
data = np.array(data).astype(float)

# 分割训练集和测试集
n_samples = len(data)
n_train = int(0.8*n_samples)
train_data = data[:n_train, :]
test_data = data[n_train:, :]

# ---------- CART 回归树算法 ----------

# 定义节点类
class Node:
    def __init__(self, split_feature, split_value, left_child, right_child, prediction):
        self.split_feature = split_feature
        self.split_value = split_value
        self.left_child = left_child
        self.right_child = right_child
        self.prediction = prediction

# 计算均方误差
def mse(y):
```

```
    return np.mean((y - np.mean(y))**2)

# 递归构建 CART 树
def build_CART_tree(X, y, min_samples_leaf=2, depth=0):
    # 如果样本数达到叶子节点要求的最小值，则直接返回叶子节点的平均值
    if len(y) <= min_samples_leaf:
        return Node(None, None, None, None, np.mean(y))

    # 如果树的深度达到最大值，则直接返回叶子节点的平均值
    if depth >= max_depth:
        return Node(None, None, None, None, np.mean(y))

    # 寻找最优的分裂特征和分裂阈值
    best_feature, best_value, best_mse = None, None, np.inf
    for i in range(X.shape[1]):
        for value in np.unique(X[:,i]):
            left_indices = X[:,i] <= value
            right_indices = X[:,i] > value
            left_mse = mse(y[left_indices])
            right_mse = mse(y[right_indices])
            mse_sum = left_mse*(sum(left_indices)/len(X)) +
right_mse*(sum(right_indices)/len(X))
            if mse_sum < best_mse:
                best_feature, best_value, best_mse = i, value, mse_sum

    # 如果没有找到满足分裂条件的节点，则直接返回叶子节点的平均值
    if best_feature is None:
        return Node(None, None, None, None, np.mean(y))

    # 根据最优的分裂特征和分裂阈值将数据集分为左右两个子树
    left_indices = X[:,best_feature] <= best_value
    right_indices = X[:,best_feature] > best_value
    left_X, left_y = X[left_indices], y[left_indices]
    right_X, right_y = X[right_indices], y[right_indices]

    # 递归构建左右两个子树
    left_child = build_CART_tree(left_X, left_y, min_samples_leaf, depth+1)
    right_child = build_CART_tree(right_X, right_y, min_samples_leaf, depth+1)

    # 返回当前节点
    return Node(best_feature, best_value, left_child, right_child, np.mean(y))

# 对新数据进行预测
def predict(node, x):
```

```
    if node.split_feature is None:
        return node.prediction
    elif x[node.split_feature] <= node.split_value:
        return predict(node.left_child, x)
    else:
        return predict(node.right_child, x)

# ------------ 执行程序 -------------

# 构建 CART 树
max_depth = 10
min_samples_leaf = 2
X_train, y_train = train_data[:,:-1], train_data[:,-1]
tree_root = build_CART_tree(X_train, y_train, min_samples_leaf)

# 在测试集上进行预测，并计算均方误差
X_test, y_test = test_data[:,:-1], test_data[:,-1]
y_pred = np.zeros(len(y_test))
for i in range(len(y_test)):
    y_pred[i] = predict(tree_root, X_test[i])
mse_test = mse(y_test - y_pred)
print("均方误差: {}".format(mse_test))

# 在测试集上进行预测
X_test, y_test = test_data[:,:-1], test_data[:,-1]
y_pred = np.zeros(len(y_test))
for i in range(len(y_test)):
    y_pred[i] = predict(tree_root, X_test[i])

# 绘制真实值和预测值比较图
plt.figure(figsize=(8, 8))
plt.scatter(range(len(y_test)), y_test, marker="o", label="True values")
plt.scatter(range(len(y_test)), y_pred, marker="x", label="Predicted values")
plt.xlabel("Test sample")
plt.ylabel("Value")
plt.title("True vs Predicted values")
plt.legend()
plt.show()
```

四、实验结果及分析

2.（1）由于 CART 算法既可以用于分类，也可以用于回归，所以本次实验采用决策树算法中的 CART 算法。将 CART 算法用于鸢尾花数据集，最后输出的准确度是 1.0，如下图所示。

```
调试控制台  输出  问题  终端                                    Python
PS D:\ALL_Project\VSCode_Files\PythonCode> & D:/Python-3.11.2/python.exe d:/ALL_Project/VSCode
_Files/PythonCode/Lab3/iris_classify.py
Accuracy: 1.0
PS D:\ALL_Project\VSCode_Files\PythonCode>
```

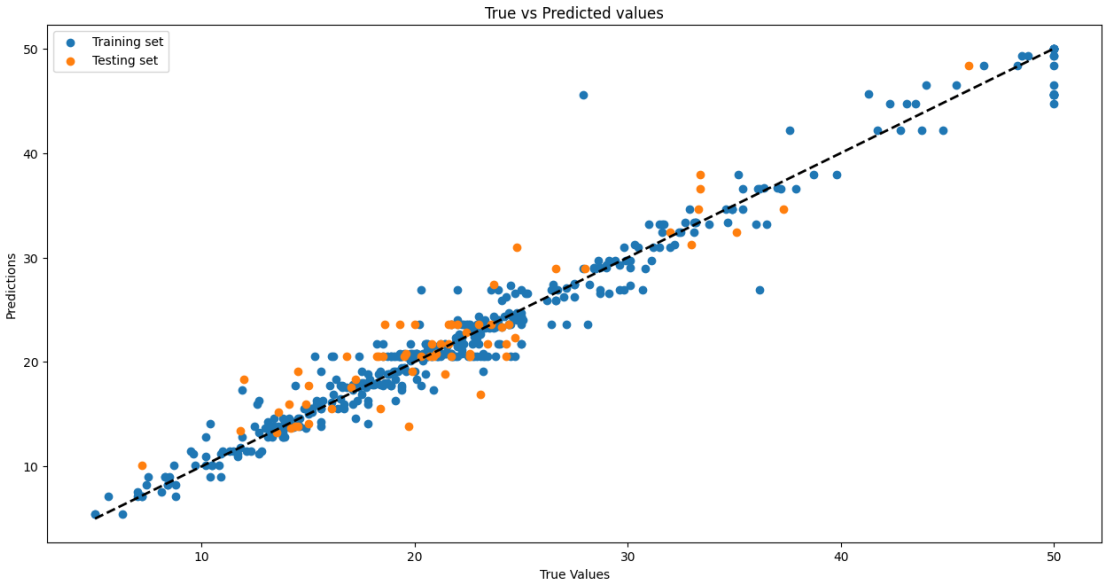
（2）将同样的算法应用到红酒数据集上，准确率显著下降，通过调整测试集的比例和决策树的最大深度，最终准确率达到了 69%。

```
调试控制台  输出  问题  终端                                    Python
PS D:\ALL_Project\VSCode_Files\PythonCode> & D:/Python-3.11.2/python.exe d:/ALL_Project/VSCode
_Files/PythonCode/Lab3/wine_classify.py
Accuracy: 0.69
PS D:\ALL_Project\VSCode_Files\PythonCode>
```

（3）使用决策树回归模型对波士顿房价进行预测的均方误差约为 6.3584，由于均方误差容易受到噪声、数据分布的影响较大，所以通过调整模型中的测试集的占比，决策树中允许分裂节点的最大深度和分裂之前分裂节点必须包含的最小样本数，可以避免过拟合，提高模型的泛化能力，进而降低均方误差。

```
调试控制台  输出  问题  终端                                    Python
PS D:\ALL_Project\VSCode_Files\PythonCode> & D:/Python-3.11.2/python.exe d:/ALL_Project/VSCode
_Files/PythonCode/Lab3/Boston_regression.py
D:\Python-3.11.2\Lib\site-packages\numpy\core\fromnumeric.py:3464: RuntimeWarning: Mean of emp
ty slice.
  return _methods._mean(a, axis=axis, dtype=dtype,
D:\Python-3.11.2\Lib\site-packages\numpy\core\_methods.py:192: RuntimeWarning: invalid value e
ncountered in scalar divide
  ret = ret.dtype.type(ret / rcount)
均方误差: 6.358397342583685
PS D:\ALL_Project\VSCode_Files\PythonCode>
```

下图将波士顿房价的真实值和预测值表示成点，并用不同的颜色标识，可以直观看出模型在训练集和测试集上的表现，以及是否存在欠拟合和过拟合等问题。

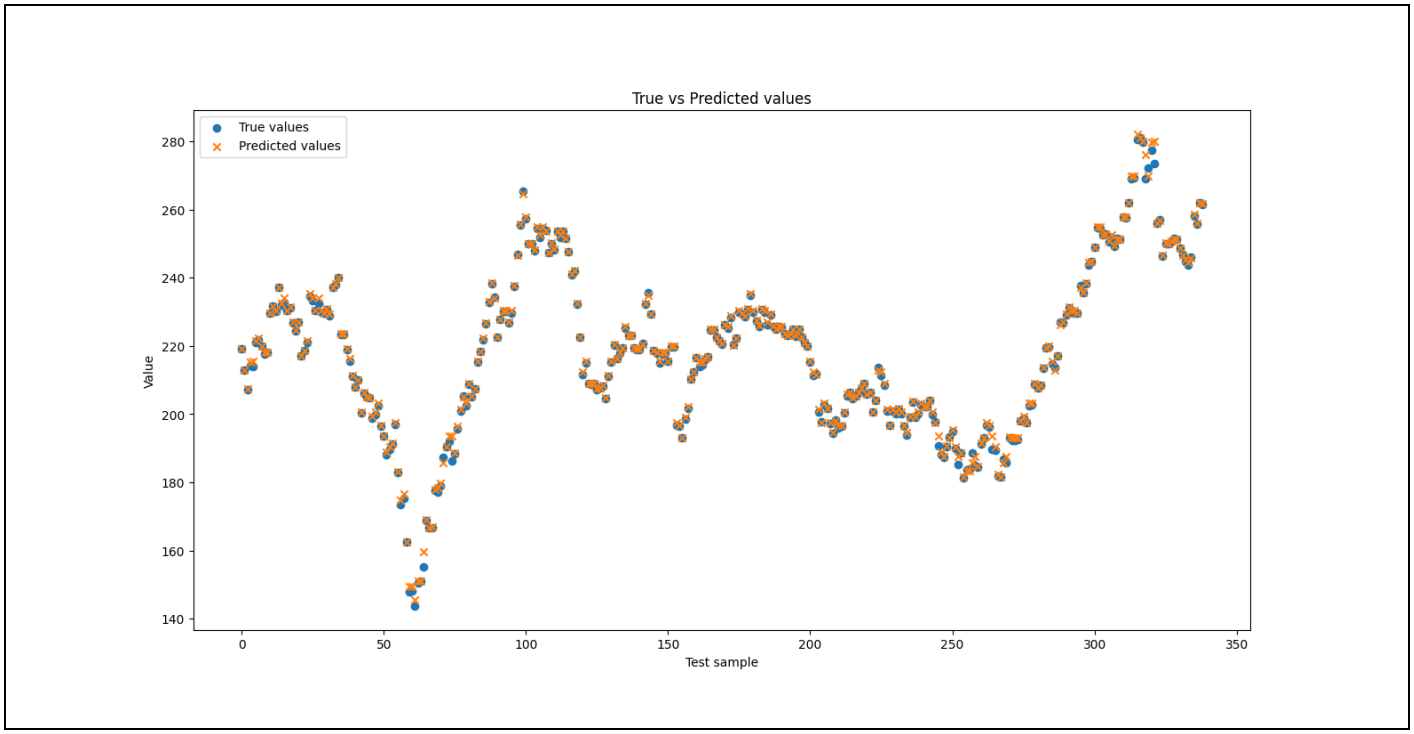


（4）同样，将决策树回归模型应用到特斯拉股票预测数据集上，发现拟合的效果很好，均方误差为 0.758，即预测结果与实际值之间平均只有 0.758 的差异。

```
调试控制台  输出  问题  终端                                    Python

PS D:\ALL_Project\VSCode_Files\PythonCode> & D:/Python-3.11.2/python.exe d:/ALL_Project/VSCode
_Files/PythonCode/Lab3/Tesla_regression.py
D:\Python-3.11.2\Lib\site-packages\numpy\core\fromnumeric.py:3464: RuntimeWarning: Mean of emp
ty slice.
  return _methods._mean(a, axis=axis, dtype=dtype,
D:\Python-3.11.2\Lib\site-packages\numpy\core\_methods.py:192: RuntimeWarning: invalid value e
ncountered in scalar divide
  ret = ret.dtype.type(ret / rcount)
均方误差: 0.7581274783502674
PS D:\ALL_Project\VSCode_Files\PythonCode>
```

从下图可以看出，预测值和真实值之间的差异较小，且在图中呈现了同样的变化趋势，说明该模型对这个测试集的拟合效果很好。


True vs Predicted values
True values
Predicted values
Value
Test sample
280
260
240
220
200
180
160
140
0
50
100
150
200
250
300
350