**机器学习应用实践（实验四） —决策树**

**工业智能2201班 刘天行 20225354**

1. **实验目的**

1、 掌握 Scikit-Learn 中决策树算法的使用，能够使用分类树和回归树算法解决实际问题。

2、掌握决策树算法中的剪枝策略对其性能的影响。

3、掌握 CART 决策树算法的原理，并具备使用 python 实现决策树算法的编程能力。

4、 掌握三种常用的类别数据编码方法

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**

**1、题目一： 采用 scikit-learn 中的 DecisionTreeClassifier 决策树对葡萄酒数据集进行预测。**

【讨论一】 模型参数对模型性能有何影响？

（ 1）不同特征选择标准（ criterion =’gini’或者’entropy’）对模型性能是否有影响？

（ 2） 不同特征划分标准（ splitter=’best’或者’random’）对模型性能是否有影响？

（ 3）尝试修改 max\_depth、 min\_samples\_leaf、 min\_samples\_split 等参数，通过树形图分析参数的作用。

【讨论二】如何确定最优的剪枝参数？

找到合适的超参数，展示调整后的模型效果。可使用学习曲线进行超参数选取，其优点是可以看到超参数对模型性能影响的趋势。

如果需要确定的超参数比较多，且超参数之间相互影响时， 可尝试使用 GridSearchCV 选择模型最优的超参数。

**2、题目二（选做） ： 使用 scikit-learn 中的 DecisionTree- -Classifier 决策树对 kddcup99 数据集进行预测。**

**3、题目三： 使用 numpy 编写的 CART 分类/回归树（选择一种即可）算法，并对 iris 数据集/california数据集进行预测。**

具体内容：

（ 1） 导入数据集。

（ 2）划分数据（分成训练集和数据集）

（ 3）训练模型（参考程序模板： cart\_numpy\_template.py）

（ 4） 输出树模型。

（ 5） 进行预测， 评估模型性能。

**拓展内容（选做）：**

（ 1）尝试加入 TN 样本数量阈值和 TG 基尼指数阈值作为终止条件。

（ 2）尝试对离散特征进行分枝

【讨论三】树递归分枝的终止条件是什么？展示对应的代码。请结合代码简述树的递归分枝的过程。

1. **实验情况**
2. **题目一：导入数据，建立模型，输出结果**

from sklearn.datasets import load\_wine

from sklearn.tree import DecisionTreeClassifier

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

from sklearn import tree

import matplotlib.pyplot as plt

# 加载葡萄酒数据

data = load\_wine()

X = data.data

y = data.target

feature\_names = data.feature\_names

target\_names = data.target\_names

# 查看数据集信息

print("数据集信息:")

print("数据量:", X.shape[0])  #X.shape[0]表示输入数据矩阵X中的样本数量或行数

print("特征数量（维度）:", X.shape[1])

print("特征名:", feature\_names)

print("标签名:", target\_names)

print("标签分布:")

for i, target\_name in enumerate(target\_names):

    print(target\_name, ":", sum(y == i))

print("数据集描述:", data.DESCR)

# 划分数据集

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# 创建决策树模型

model = DecisionTreeClassifier()

# 训练模型

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

# 输出特征重要程度

importances = model.feature\_importances\_

print("特征重要程度(占比):")

#使用zip函数，循环将同时迭代feature\_names和importances列表中的元素

for feature\_name, importance in zip(feature\_names, importances):

    print(feature\_name, ":", importance)

# 计算分类准确率

accuracy = model.score(X\_test, y\_test)

print("\n分类准确率:", accuracy)

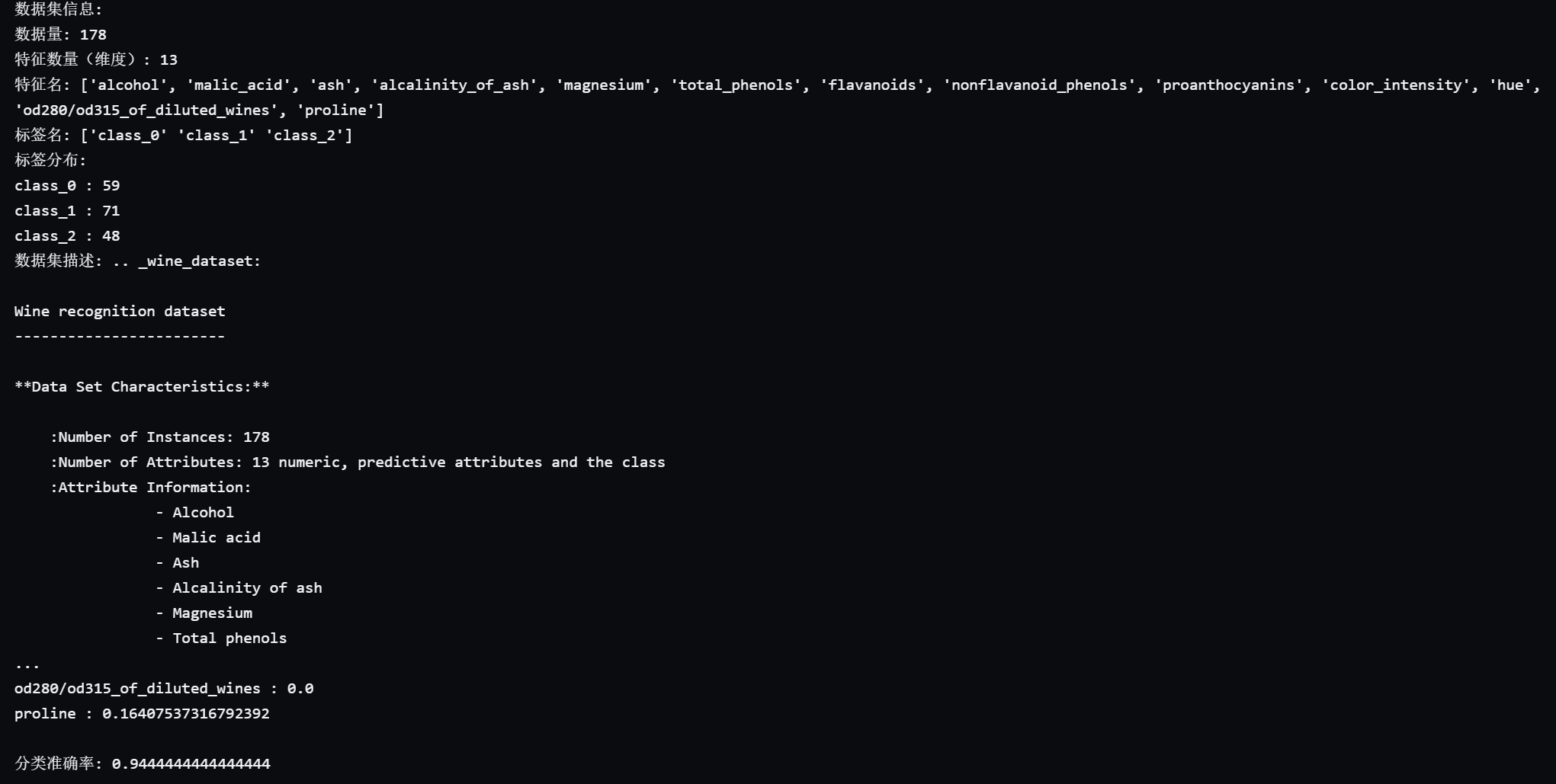
# 绘制决策树

plt.figure(figsize=(10, 6))

tree.plot\_tree(model, feature\_names=feature\_names, class\_names=target\_names, filled=True)

plt.show()

**得到运行结果和决策树如下：**



**日程表

描述已自动生成**

**【讨论一】模型参数对模型性能有哪些影响？**

**为了探究各个模型参数对模型性能的影响，我们采用分别控制变量，遍历参数取值分别训练模型的方式来进行尝试，根据这个思想，我们写出如下的代码：**

# 不同特征选择，这里使用gini和entropy两种标准

model1 = DecisionTreeClassifier(criterion='gini')

model2 = DecisionTreeClassifier(criterion='entropy')

# 训练模型

model1.fit(X\_train, y\_train)

model2.fit(X\_train, y\_train)

# 输出特征重要性

importances1 = model1.feature\_importances\_

importances2 = model2.feature\_importances\_

print("\n特征重要性 (criterion=gini):")

for feature\_name, importance in zip(feature\_names, importances1):

    print(feature\_name, ":", importance)

print("\n特征重要性 (criterion=entropy):")

for feature\_name, importance in zip(feature\_names, importances2):

    print(feature\_name, ":", importance)

# 计算测试集上的准确率

accuracy1 = model1.score(X\_test, y\_test)

accuracy2 = model2.score(X\_test, y\_test)

print("\n准确率(criterion=gini):", accuracy1)

print("准确率(criterion=entropy):", accuracy2)

#用不同特征划分标准建立决策树模型

model3 = DecisionTreeClassifier(splitter='best')

model4 = DecisionTreeClassifier(splitter='random')

# 训练模型

model3.fit(X\_train, y\_train)

model4.fit(X\_train, y\_train)

# 不同特征划分的决策树绘制

plt.figure(figsize=(12, 6))

tree.plot\_tree(model3, feature\_names=feature\_names, class\_names=target\_names, filled=True)

plt.title("Decision Tree (splitter=best)")

plt.show()

plt.figure(figsize=(12, 6))

tree.plot\_tree(model4, feature\_names=feature\_names, class\_names=target\_names, filled=True)

plt.title("Decision Tree (splitter=random)")

plt.show()

# 建立不同参数的模型（控制单一变量）

#探究max\_depth影响，保持min\_samples\_leaf=2, min\_samples\_split=5

# 划分训练集和测试集

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# 遍历不同的max\_depth值

max\_depth\_values = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

results = []

for max\_depth in max\_depth\_values:

    # 创建决策树模型

    model = DecisionTreeClassifier(max\_depth=max\_depth,min\_samples\_leaf=2, min\_samples\_split=5)

    # 在训练集上训练模型

    model.fit(X\_train, y\_train)

    plt.figure(figsize=(12, 6))

    tree.plot\_tree(model, feature\_names=feature\_names, class\_names=target\_names, filled=True)

    plt.title('Decision Tree with max\_depth=%d'%max\_depth)

    plt.show()

#探究min\_samples\_leaf影响，保持max\_depth=3,min\_samples\_split=5

# 遍历不同的min\_samples\_leaf值

min\_samples\_leaf\_values = [1, 3, 5, 7, 9]

results2 = []

for min\_samples\_leaf in min\_samples\_leaf\_values:

    # 创建决策树模型

    model = DecisionTreeClassifier(max\_depth=3,min\_samples\_leaf=min\_samples\_leaf,min\_samples\_split=5)

    model.fit(X\_train, y\_train)

    plt.figure(figsize=(12, 6))

    tree.plot\_tree(model, feature\_names=feature\_names, class\_names=target\_names, filled=True)

    plt.title('Decision Tree with min\_samples\_leaf=%d'%min\_samples\_leaf)

    plt.show()

#探究min\_samples\_split影响，保持max\_depth=3,min\_samples\_leaf=2

# 遍历不同的min\_samples\_split值

min\_samples\_split\_values = [2,5,10]

results3 = []

for min\_samples\_split in min\_samples\_split\_values:

    # 创建决策树模型

    model = DecisionTreeClassifier(max\_depth=3,min\_samples\_leaf=2,min\_samples\_split=min\_samples\_split)

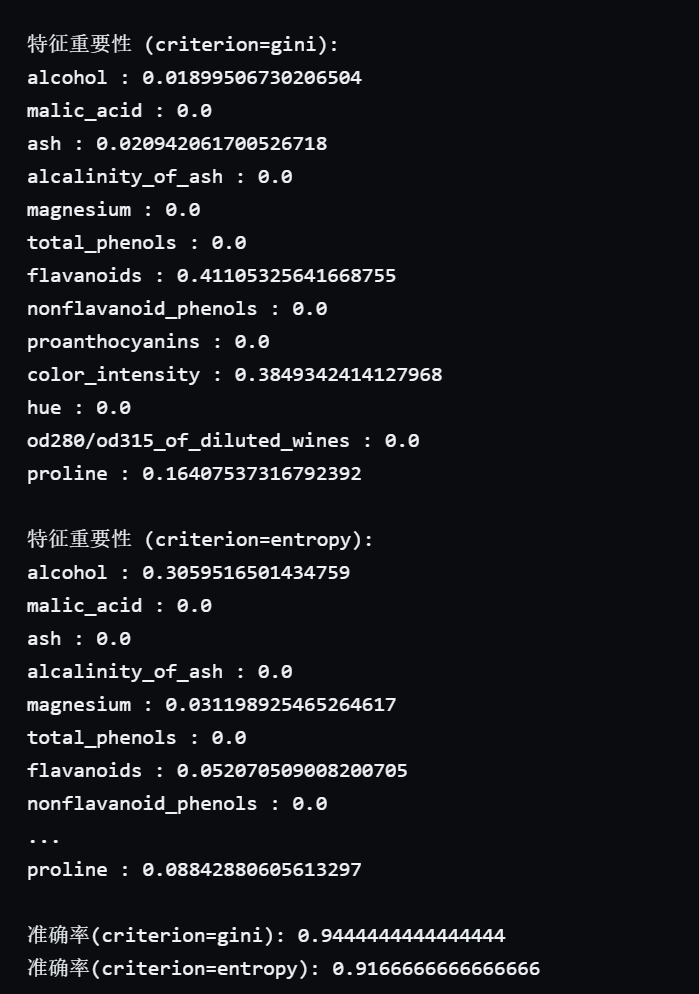
    model.fit(X\_train, y\_train)

    plt.figure(figsize=(12, 6))

    tree.plot\_tree(model, feature\_names=feature\_names, class\_names=target\_names, filled=True)

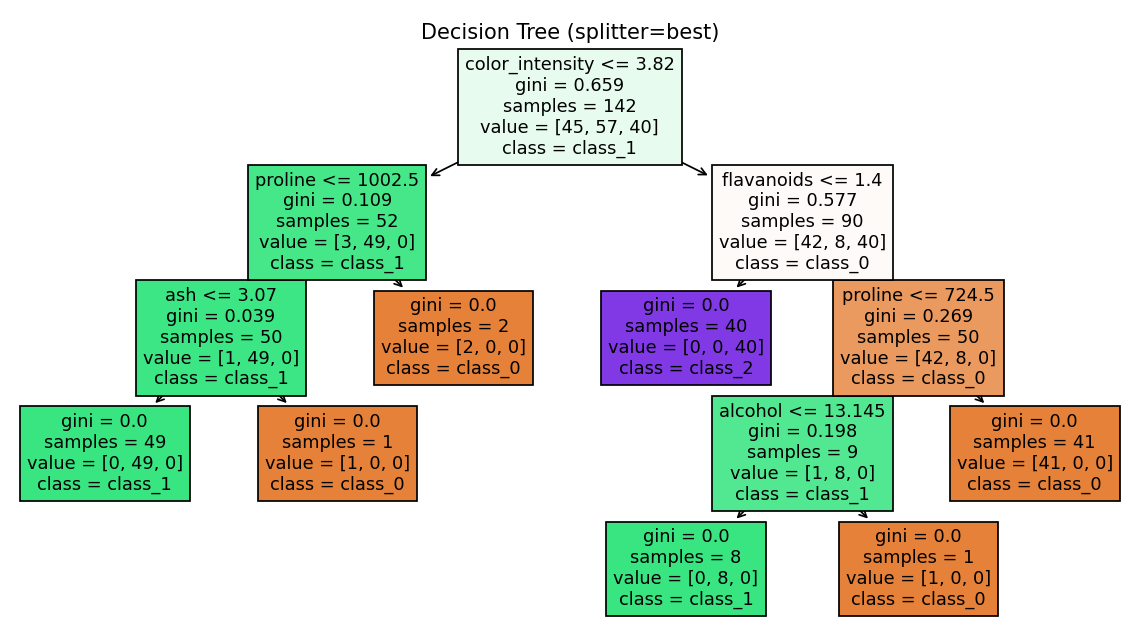
    plt.title('Decision Tree with min\_samples\_split=%d'%min\_samples\_split)

    plt.show()

**运行结果如下：**

1.对比不同特征选择标准（criterion =’gini’或者’entropy’），可以看出：采用分裂准则'gini'比'entropy'效果更好，模型准确率更高，泛化性能更强。

1. 对比不同特征划分标准（splitter=’best’或者’random’）：



图示

描述已自动生成

可以看出random所得到的决策树比best更加复杂，使用'best'的划分标准的决策树模型具有更简洁，说明'best'的划分标准对模型性能具有积极影响。

3.改变决策树的最大深度max\_depth（1-10）

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

增加最大深度可能会导致模型更复杂，可能会导致容易过拟合训练数据。

4.改变叶子节点上所需的最小样本数量min\_samples\_leaf的值（1, 3, 5, 7, 9）

|  |  |
| --- | --- |
| 图片包含 图示  描述已自动生成 | 图片包含 日程表  描述已自动生成 |
| 图示  描述已自动生成 | 图片包含 图示  描述已自动生成 |
| 图示  描述已自动生成 |  |

增加叶节点的最小样本数可以导致树的分支较少，减少过拟合的风险，但可能会导致模型的偏差增加。

5.改变内部节点分裂所需的最小样本数量min\_samples\_split的值（2,5,10）：

|  |
| --- |
| 图片包含 日程表  描述已自动生成 |
| 图片包含 日程表  描述已自动生成 |
| 图片包含 日程表  描述已自动生成 |

增加节点分裂所需的最小样本数可以限制决策树的生长，防止过拟合，但可能会导致欠拟合。

**【讨论二】如何确定最优的剪枝策略？**

**为了解决超参数寻优问题，我们采取学习曲线和实验指导中提到的GridSearchCV方法，依然是基于遍历的思想，设置超参数范围，使用GridSearchCV来实现超参数自动寻优，写出下面的代码**

from sklearn.model\_selection import validation\_curve

import numpy as np

# 设置超参数范围

param\_range = np.arange(1, 11)  # max\_depth from 1 to 10

train\_scores, test\_scores = validation\_curve(

    DecisionTreeClassifier(min\_samples\_leaf=2, min\_samples\_split=5),

    X, y,

    param\_name="max\_depth",

    param\_range=param\_range,

    cv=5,

    scoring="accuracy",

    n\_jobs=-1

)

# 计算平均和标准差

train\_scores\_mean = np.mean(train\_scores, axis=1)

train\_scores\_std = np.std(train\_scores, axis=1)

test\_scores\_mean = np.mean(test\_scores, axis=1)

test\_scores\_std = np.std(test\_scores, axis=1)

# 绘制学习曲线

plt.fill\_between(param\_range, train\_scores\_mean - train\_scores\_std,

                 train\_scores\_mean + train\_scores\_std, alpha=0.1, color="r")

plt.fill\_between(param\_range, test\_scores\_mean - test\_scores\_std,

                 test\_scores\_mean + test\_scores\_std, alpha=0.1, color="g")

plt.plot(param\_range, train\_scores\_mean, 'o-', color="r",

         label="Training score")

plt.plot(param\_range, test\_scores\_mean, 'o-', color="g",

         label="Cross-validation score")

plt.title("Learning Curve for max\_depth")

plt.xlabel('max\_depth')

plt.ylabel('Accuracy')

plt.legend(loc="best")

plt.show()

from sklearn.model\_selection import GridSearchCV

# 设置超参数网格

param\_grid = {

    'max\_depth': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],

    'min\_samples\_leaf': [1, 2, 3, 4, 5],

    'min\_samples\_split': [2, 5, 10, 20]

}

# 创建GridSearchCV对象

grid\_search = GridSearchCV(DecisionTreeClassifier(), param\_grid, cv=5, scoring='accuracy', n\_jobs=-1)

grid\_search.fit(X\_train, y\_train)

# 输出最佳参数和对应的准确率

print("Best parameters:", grid\_search.best\_params\_)

print("Best cross-validation score: {:.2f}".format(grid\_search.best\_score\_))

# 使用最优参数重新训练模型

best\_model = grid\_search.best\_estimator\_

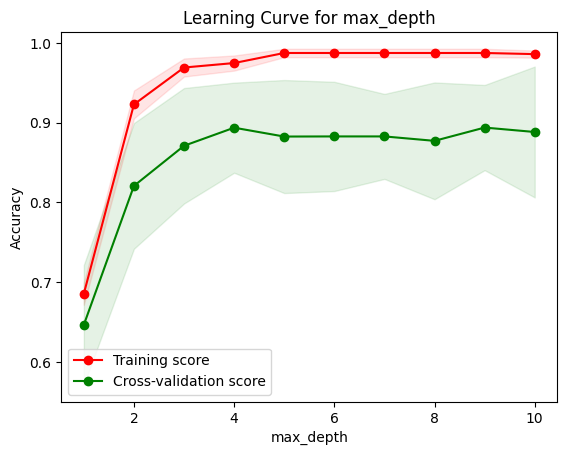
plt.figure(figsize=(12, 6))

tree.plot\_tree(best\_model, feature\_names=feature\_names, class\_names=target\_names, filled=True)

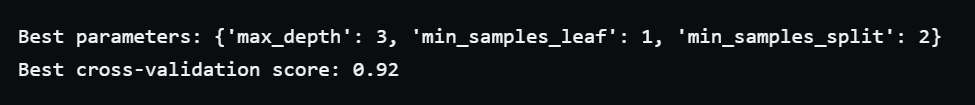
plt.title("Optimized Decision Tree")

plt.show()

**得到结果如下：**

**学习曲线：**

**得到的最优参数：**



**最优参数对应的决策树：**

**图片包含 日程表

描述已自动生成**

1. **题目二：**

**首先导入数据并进行编码：**

#导入所需的库和模块

from sklearn.datasets import fetch\_kddcup99

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

from sklearn.preprocessing import LabelEncoder

from sklearn.tree import DecisionTreeRegressor

from sklearn.metrics import mean\_squared\_error

# 获取数据集，选择获取总数据量的 10%

data = fetch\_kddcup99(percent10=True)

# 将特征和标签分开

X = data.data

y = data.target

# 使用LabelEncoder对文本特征进行编码，将其转换为数值类型

label\_encoder = LabelEncoder()

X[:, 1] = label\_encoder.fit\_transform(X[:, 1])

X[:, 2] = label\_encoder.fit\_transform(X[:, 2])

X[:, 3] = label\_encoder.fit\_transform(X[:, 3])

# 将标签进行编码

y = label\_encoder.fit\_transform(y)

#划分数据集

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

**这里的LabelEncoder 是一个用于将文本标签编码为数值标签的类。LabelEncoder类通过将每个文本标签映射到一个唯一的整数值来实现这一转换。使用LabelEncoder类，将每个不同的文本标签映射到一个唯一的整数值。**

**在使用机器学习算法之前，将文本特征转换为数值类型，因为大多数机器学习算法都只能处理数值型数据。通过这种编码过程，我们可以为每个不同的文本标签赋予一个唯一的整数值，以便算法能够对其进行计算。**

**之后，建立模型并使用网格搜索完成超参数的寻优**

# 将标签进行编码

y = label\_encoder.fit\_transform(y)

#划分数据集

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

#创建并训练决策树模型

model = DecisionTreeRegressor()

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

#进行预测并评估模型

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

mse = mean\_squared\_error(y\_test, y\_pred)

print('预测：\n',y\_pred)

print("Mean Squared Error:", mse)

#对模型进行参数调优

from sklearn.model\_selection import GridSearchCV

# 定义参数网格

param\_grid = {

    'max\_depth': [None, 5, 10],

    'min\_samples\_split': [2, 5, 10]

              }

# 执行网格搜索

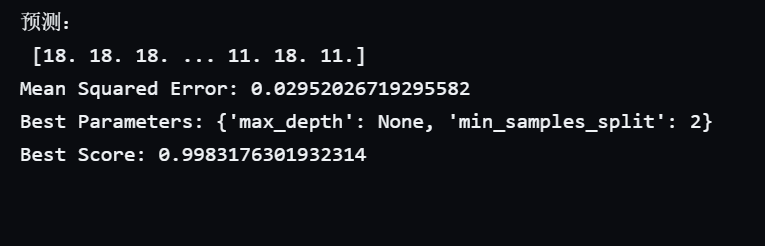
grid\_search = GridSearchCV(estimator=model, param\_grid=param\_grid, cv=10, n\_jobs=-1)

grid\_search.fit(X\_train, y\_train)

# 输出最佳参数组合和对应的模型评分

print("Best Parameters:", grid\_search.best\_params\_)

print("Best Score:", grid\_search.best\_score\_)

**得到如下的输出**

1. **题目三：**

**代码实现：**

import numpy as np

from sklearn.datasets import load\_iris

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

iris = load\_iris() # 加载数据集

X\_train, X\_test, y\_train, y\_test = train\_test\_split(iris.data, iris.target, test\_size=0.2, random\_state=42) # 划分数据集

class CART:

    def \_\_init\_\_(self, max\_depth=5, min\_samples\_split=2, min\_samples\_leaf=1):

        self.max\_depth = max\_depth # 最大深度

        self.min\_samples\_split = min\_samples\_split # 最小分割样本数

        self.min\_samples\_leaf = min\_samples\_leaf # 最小叶子节点样本数

    def fit(self, X, y):# 训练

        self.tree = self.build\_tree(X, y) # 构建决策树

    def predict(self, X):

        return np.array([self.traverse\_tree(x, self.tree) for x in X]) # 预测

    def build\_tree(self, X, y, depth=0):# 构建决策树，返回值是字典决策树

        n\_samples, n\_features = X.shape # 样本数和特征数

        n\_labels = len(np.unique(y)) # 标签数

        if depth >= self.max\_depth or n\_labels == 1 or n\_samples < self.min\_samples\_split:

            return np.argmax(np.bincount(y)) # 返回出现次数最多的标签

        feature\_idxs = np.random.choice(n\_features, n\_features, replace=False) # 随机选择特征

        best\_feature, best\_threshold = None, None

        best\_gini = np.inf

        for idx in feature\_idxs:

            thresholds = np.unique(X[:, idx]) # 特征的唯一值

            for threshold in thresholds:

                left\_idxs = np.where(X[:, idx] <= threshold)[0] # 左子树的索引

                right\_idxs = np.where(X[:, idx] > threshold)[0] # 右子树的索引

                if len(left\_idxs) == 0 or len(right\_idxs) == 0:

                    continue

                left\_gini = self.gini(y[left\_idxs]) # 左子树的基尼指数

                right\_gini = self.gini(y[right\_idxs]) # 右子树的基尼指数

                gini = (len(left\_idxs) / n\_samples) \* left\_gini + (len(right\_idxs) / n\_samples) \* right\_gini # 加权平均基尼指数

                if gini < best\_gini:

                    best\_feature = idx # 最佳特征

                    best\_threshold = threshold # 最佳阈值

                    best\_gini = gini # 最佳基尼指数

        left\_idxs = np.where(X[:, best\_feature] <= best\_threshold)[0] # 左子树的索引

        right\_idxs = np.where(X[:, best\_feature] > best\_threshold)[0] # 右子树的索引

        if len(left\_idxs) < self.min\_samples\_leaf or len(right\_idxs) < self.min\_samples\_leaf:

            return np.argmax(np.bincount(y)) # 返回出现次数最多的标签

        left\_tree = self.build\_tree(X[left\_idxs], y[left\_idxs], depth+1) # 构建左子树

        right\_tree = self.build\_tree(X[right\_idxs], y[right\_idxs], depth+1) # 构建右子树

        return {'feature': best\_feature, 'threshold': best\_threshold, 'left': left\_tree, 'right': right\_tree} # 返回决策树

    # 遍历决策树，返回值是预测的标签

    def traverse\_tree(self, x, tree):

        if type(tree) != dict:

            return tree

        if x[tree['feature']] <= tree['threshold']:

            return self.traverse\_tree(x, tree['left']) # 遍历左子树

        else:

            return self.traverse\_tree(x, tree['right']) # 遍历右子树

    # 计算基尼指数

    def gini(self, y):

        \_, counts = np.unique(y, return\_counts=True) # 标签的唯一值和出现次数

        proportions = counts / len(y) # 标签的比例

        return 1 - np.sum(proportions\*\*2) # 基尼指数

    # 打印决策树

    def print\_tree\_recursive(self, tree, depth=0):

        if isinstance(tree, dict):

            for key, value in tree.items():

                print("|   " \* depth + "|---" + str(key) + ": ", end="")

                if isinstance(value, dict):

                    print()

                    self.print\_tree\_recursive(value, depth + 1)

                else:

                    print(value)

        else:

            print("|   " \* depth + "|---" + str(tree))

cart = CART() # 创建决策树对象

cart.fit(X\_train, y\_train) # 训练决策树

# 输出树模型

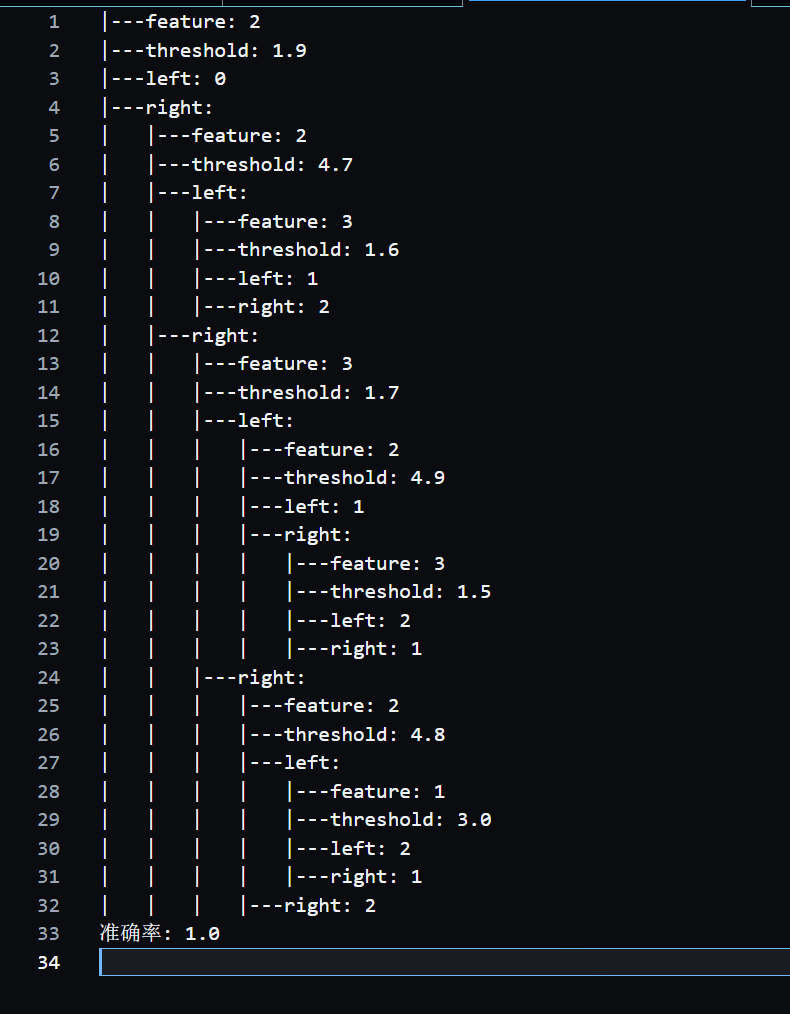
cart.print\_tree\_recursive(cart.tree)

y\_pred = cart.predict(X\_test) # 预测

accuracy = np.sum(y\_pred == y\_test) / len(y\_test) # 计算准确率

print('准确率:', accuracy) # 打印准确率

**得到如下的决策树：**



**尝试加入 TN样本数量阈值和 TG基尼指数阈值作为终止条件；尝试对离散特征进行分枝。**

import numpy as np

from sklearn.datasets import load\_iris

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

from collections import Counter

class CART:

    def \_\_init\_\_(self, max\_depth=5, min\_samples\_split=2, min\_samples\_leaf=1, tn\_threshold=0, tg\_threshold=0):

        self.max\_depth = max\_depth

        self.min\_samples\_split = min\_samples\_split

        self.min\_samples\_leaf = min\_samples\_leaf

        self.tn\_threshold = tn\_threshold  # TN 样本数量阈值

        self.tg\_threshold = tg\_threshold  # TG 基尼指数阈值

    def fit(self, X, y):

        self.tree = self.build\_tree(X, y)

    def predict(self, X):

        return np.array([self.traverse\_tree(x, self.tree) for x in X])

    def build\_tree(self, X, y, depth=0):

        n\_samples, n\_features = X.shape

        n\_labels = len(np.unique(y))

        if (

            depth >= self.max\_depth

            or n\_labels == 1

            or n\_samples < self.min\_samples\_split

            or Counter(y)[0] >= self.tn\_threshold

            or self.gini(y) <= self.tg\_threshold

        ):

            return np.argmax(np.bincount(y))

        best\_feature, best\_threshold = None, None

        best\_gini = np.inf

        for idx in range(n\_features):

            unique\_values = np.unique(X[:, idx])

            if len(unique\_values) <= 1:  # Skip if only one unique value

                continue

            for threshold in unique\_values:

                left\_idxs = np.where(X[:, idx] <= threshold)[0]

                right\_idxs = np.where(X[:, idx] > threshold)[0]

                if len(left\_idxs) == 0 or len(right\_idxs) == 0:

                    continue

                left\_gini = self.gini(y[left\_idxs])

                right\_gini = self.gini(y[right\_idxs])

                gini = (len(left\_idxs) / n\_samples) \* left\_gini + (len(right\_idxs) / n\_samples) \* right\_gini

                if gini < best\_gini:

                    best\_feature = idx

                    best\_threshold = threshold

                    best\_gini = gini

        if best\_feature is None:  # No valid split found

            return np.argmax(np.bincount(y))

        left\_idxs = np.where(X[:, best\_feature] <= best\_threshold)[0]

        right\_idxs = np.where(X[:, best\_feature] > best\_threshold)[0]

        if (

            len(left\_idxs) < self.min\_samples\_leaf

            or len(right\_idxs) < self.min\_samples\_leaf

            or Counter(y)[0] >= self.tn\_threshold

            or self.gini(y) <= self.tg\_threshold

        ):

            return np.argmax(np.bincount(y))

        left\_tree = self.build\_tree(X[left\_idxs], y[left\_idxs], depth + 1)

        right\_tree = self.build\_tree(X[right\_idxs], y[right\_idxs], depth + 1)

        return {"feature": best\_feature, "threshold": best\_threshold, "left": left\_tree, "right": right\_tree}

    def traverse\_tree(self, x, tree):

        if isinstance(tree, dict):

            if x[tree['feature']] <= tree['threshold']:

                return self.traverse\_tree(x, tree['left'])  # Traverse left subtree

            else:

                return self.traverse\_tree(x, tree['right'])  # Traverse right subtree

        else:

            return tree

    def gini(self, y):

        \_, counts = np.unique(y, return\_counts=True)

        proportions = counts / len(y)

        return 1 - np.sum(proportions\*\*2)

    def print\_tree\_recursive(self, tree, depth=0):

        if isinstance(tree, dict):

            for key, value in tree.items():

                print("|   " \* depth + "|---" + str(key) + ": ", end="")

                if isinstance(value, dict):

                    print()

                    self.print\_tree\_recursive(value, depth + 1)

                else:

                    print(value)

        else:

            print("|   " \* depth + "|---" + str(tree))

iris = load\_iris()

X\_train, X\_test, y\_train, y\_test = train\_test\_split(iris.data, iris.target, test\_size=0.2, random\_state=42)

cart = CART(max\_depth=5, min\_samples\_split=2, min\_samples\_leaf=1, tn\_threshold=0, tg\_threshold=0)

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

# Output the tree model

cart.print\_tree\_recursive(cart.tree)

y\_pred = cart.predict(X\_test)

accuracy = np.sum(y\_pred == y\_test) / len(y\_test)

print('准确率:', accuracy)

**运行结果：**

**{'index': 2, 'value': 3.0, 'left': {'index': 0, 'value': 4.6, 'left': {'index': 0, 'value': 4.4, 'left': 0.0, 'right': {'index': 0, 'value': 4.4, 'left': 0.0, 'right': 0.0}}, 'right': {'index': 0, 'value': 4.6, 'left': 0.0, 'right': 0.0}}, 'right': {'index': 2, 'value': 4.8, 'left': {'index': 3, 'value': 1.7, 'left': {'index': 0, 'value': 6.7, 'left': {'index': 0, 'value': 6.4, 'left': 1.0, 'right': 1.0}, 'right': {'index': 0, 'value': 6.7, 'left': 1.0, 'right': 1.0}}, 'right': 2.0}, 'right': {'index': 3, 'value': 1.8, 'left': {'index': 2, 'value': 5.0, 'left': {'index': 0, 'value': 6.3, 'left': 1.0, 'right': 1.0}, 'right': {'index': 3, 'value': 1.6, 'left': 2.0, 'right': 1.0}}, 'right': {'index': 2, 'value': 4.9, 'left': {'index': 0, 'value': 6.0, 'left': 1.0, 'right': 2.0}, 'right': {'index': 0, 'value': 6.3, 'left': 2.0, 'right': 2.0}}}}}**

**准确率: 1.0**

**【讨论三】树递归分枝的终止条件是什么？展示对应的代码。请结合代码简述树的递归分枝的过程。**

**决策树递归分裂的终止条件包括以下几种情况：**

**达到最大深度：当节点的深度达到了预设的最大深度 max\_depth 时，停止分裂，并将该节点标记为叶子节点。**

if depth >= self.max\_depth:

    return np.argmax(np.bincount(y))

**样本数量不足：当节点中的样本数少于预设的最小分割样本数 min\_samples\_split 时，停止分裂，并将该节点标记为叶子节点。**

if n\_samples < self.min\_samples\_split:

    return np.argmax(np.bincount(y))

**叶子节点样本数不足：如果分裂后的左子树或右子树中的样本数少于预设的最小叶子节点样本数 min\_samples\_leaf，则停止分裂，并将该节点标记为叶子节点。**

if len(left\_idxs) < self.min\_samples\_leaf or len(right\_idxs) < self.min\_samples\_leaf:

    return np.argmax(np.bincount(y))

**负类样本数量超标：当节点中负类（True Negative）样本数量超过设定的阈值 tn\_threshold 时，停止分裂，并将该节点标记为叶子节点。**

if Counter(y)[0] >= self.tn\_threshold:

    return np.argmax(np.bincount(y))

**基尼指数小于等于阈值：当节点的基尼指数（Gini Index）小于等于设定的阈值 tg\_threshold 时，停止分裂，并将该节点标记为叶子节点。**

if self.gini(y) <= self.tg\_threshold:

    return np.argmax(np.bincount(y))

**在决策树类的 build\_tree 方法中，增加了上述终止条件，通过相应的条件语句实现这些逻辑。当任何一个终止条件满足时，决策树将停止分裂，且不再生成新的子节点。最终，这些终止条件共同构建了一个具备终止机制的决策树模型。**