**课后作业四：决策树**

任务：基于决策树构建机器学习模型，根据乘客的特征预测其在 Titanic 沉船事件中是否幸存。

数据集：Titanic数据集中乘客的特征包含客舱等级、性别、年龄、在 Titanic 号上的同伴/配偶数量、船票编号、票价等。对于每一个乘客都包含了其是否在Titanic灾难中生还的信息（Survived），作为真实值标签。

**普通实现：**

1.导入必要的库

import graphviz #一个开源的图形可视化工具，可以用来生成和展示图形结构

import pandas as pd

import pydotplus #pydotplus 是一个图形库，基于 pydot 和 Graphviz，允许创建和操作 DOT 文件

from IPython.display import Image #一个 IPython 模块，用于在 Jupyter Notebook 中嵌入和显示图像。

from sklearn.metrics import accuracy\_score #用于计算分类模型的准确率

from sklearn.model\_selection import GridSearchCV, train\_test\_split #优化模型的超参数

from sklearn.tree import DecisionTreeClassifier, export\_graphviz #用于创建和训练决策树分类模型

2.加载数据

# 加载数据

train\_data = pd.read\_csv('D:\\dataenclorse\\forth\\train.csv')

test\_data = pd.read\_csv('D:\\dataenclorse\\forth\\test.csv')

submission = pd.read\_csv('D:\\dataenclorse\\forth\\submission.csv')

3.定义预处理函数

def preprocess\_data(data):

    # 处理缺失

    data['Age'].fillna(data['Age'].median(), inplace=True)

    data['Embarked'].fillna(data['Embarked'].mode()[0], inplace=True)

    data['Fare'].fillna(data['Fare'].median(), inplace=True)

    # 将性别和登船港口转换为数值

    data['Sex'] = data['Sex'].map({'male': 0, 'female': 1})

    data['Embarked'] = data['Embarked'].map({'C': 0, 'Q': 1, 'S': 2})

    # 删除不必要的列

    data.drop(['Name', 'Ticket', 'Cabin', 'PassengerId'], axis=1, inplace=True)

    return data

4.数据预处理

# 预处理数据

train\_data = preprocess\_data(train\_data)

test\_data = preprocess\_data(test\_data)

# 特征和标签

X = train\_data.drop('Survived', axis=1)

y = train\_data['Survived']

# 分割数据集为训练集和验证集

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

5.定义参数网格

# 定义参数网格

param\_grid = {

    'max\_depth': [None, 10, 20, 30, 40, 50],

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

    'min\_samples\_leaf': [1, 5, 10],

    'max\_features': [None, 'sqrt', 'log2']

}

6.创建决策树分类器并验证准确率

# 创建决策树分类器

clf = DecisionTreeClassifier(random\_state=42)

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

# 验证模型

y\_pred = clf.predict(X\_val)

accuracy = accuracy\_score(y\_val, y\_pred)

print(f'Validation Accuracy: {accuracy:.4f}')

7.对测试集进行预测并填入submission.csv

# 对测试集进行预测

test\_pred = clf.predict(test\_data)

# 将预测结果填入submission.csv

submission['Survived'] = test\_pred

# 保存结果到submission.csv

submission.to\_csv('D:\\dataenclorse\\forth\\submission.csv', index=False)

可视化决策树并保存为图片

dot\_data = export\_graphviz(

    clf,

    out\_file=None,

    feature\_names=X.columns,

    class\_names=['Not Survived', 'Survived'],

    filled=True,

    rounded=True,

    special\_characters=True

)

8.可视化决策树

dot\_data = export\_graphviz(

    clf,

    out\_file=None,

    feature\_names=X.columns,

    class\_names=['Not Survived', 'Survived'],

    filled=True,

    rounded=True,

    special\_characters=True

)

graph = pydotplus.graph\_from\_dot\_data(dot\_data)

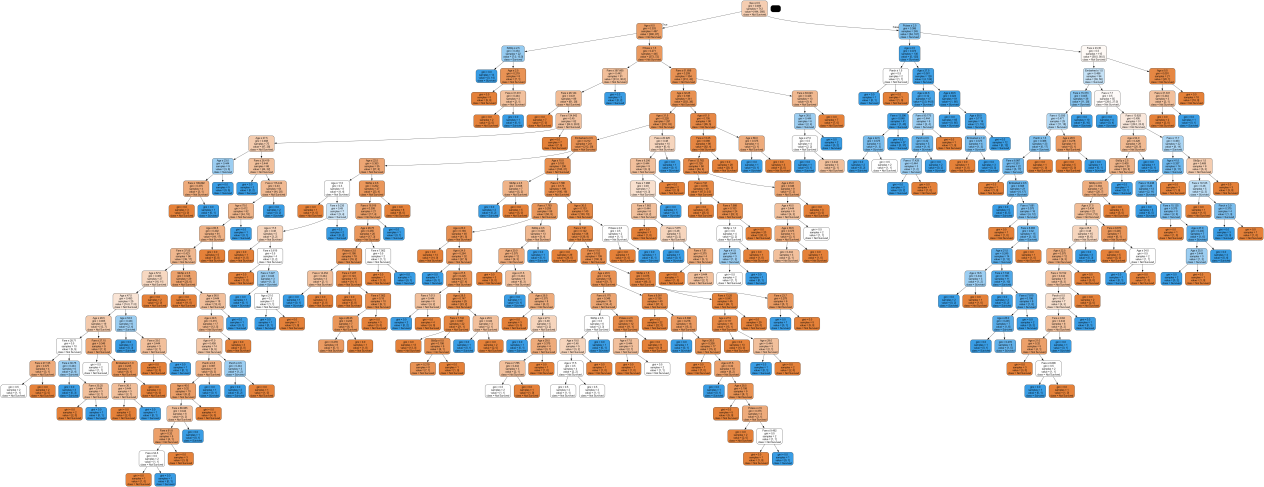
graph.write\_png('D:\\dataenclorse\\forth\\decision\_tree.png')

9.结果如下：

准确率为0.7821，不是很理想



决策树如下（枝叶茂密，决策拖泥带水）：



**优化实现：**

1.使用网格搜索进行超参数调优

grid\_search = GridSearchCV(clf, param\_grid, cv=5, scoring='accuracy')

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

2.打印最佳参数和最佳得分

best\_params = grid\_search.best\_params\_

best\_score = grid\_search.best\_score\_

print(f'Best Parameters: {best\_params}')

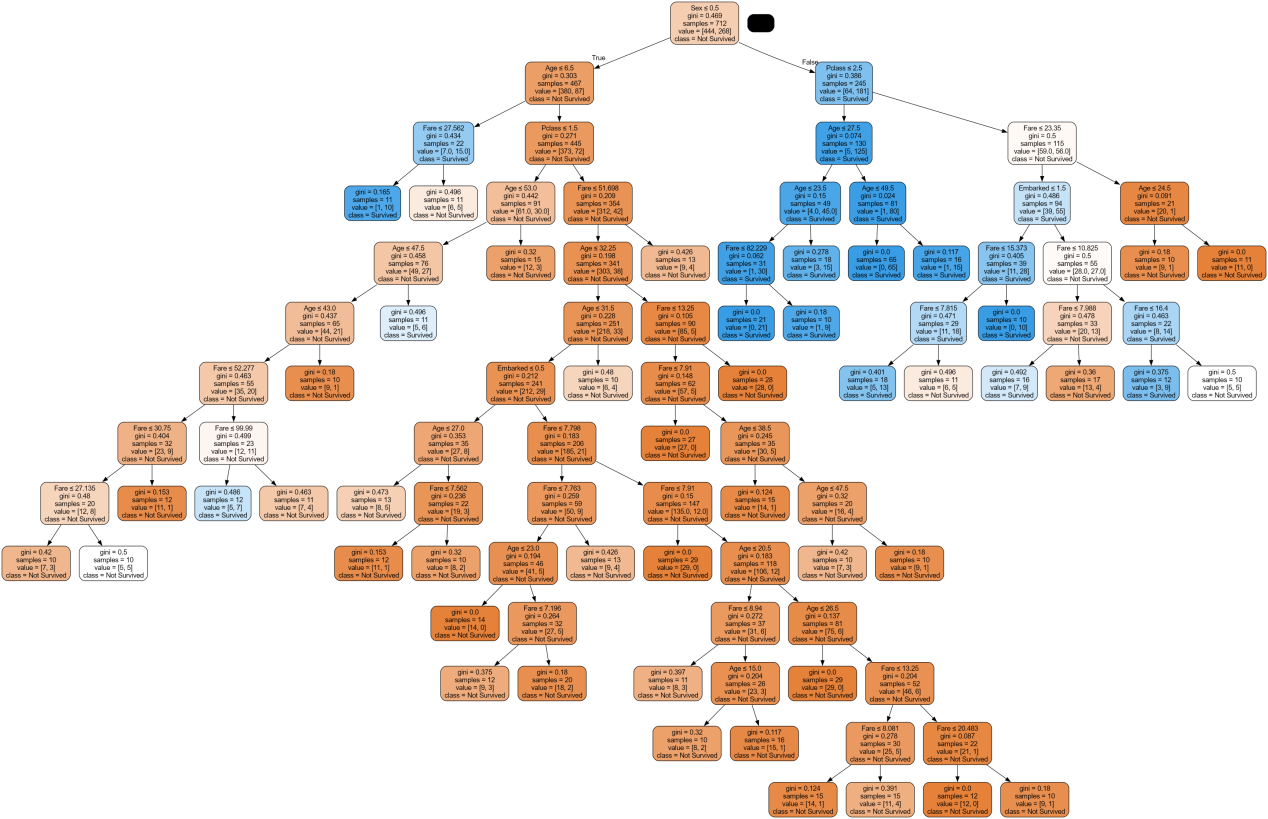
print(f'Best Score: {best\_score:.4f}')

3.优化结果如下：

最高分为：0.8104，准确率为0.7989，有了部分提高



决策树如下（相比普通的决策树，优化后的决策树更加简洁高效）：



Submission.csv如下（只展示部分）：

