

# 中北大学软件学院

## 实 验 报 告

专 业:	软件工程
方 向:	人工智能
课程名称:	机器学习实践
班 级:	22130418
学 号:	2213041835、2213041816
姓 名:	吕炳荣、刘晓峰
辅导教师:	程晓鼎

2024 年 3 月 制

成绩： \_\_\_\_\_

实验时间	2024 年 3 日 30 时 14 至 18 时	学时数	4 学时
------	---------------------------	-----	------

### 1. 实验名称

空气质量等级的决策树分类预测

### 2. 实验目的

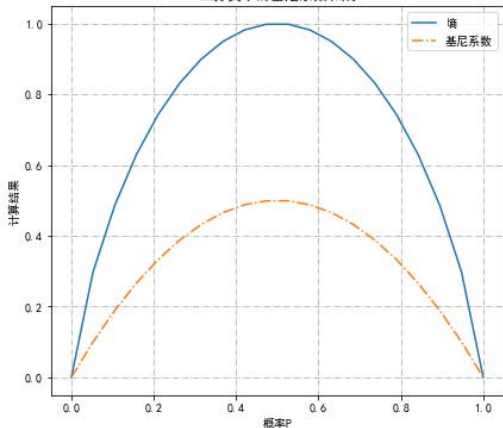
熟悉和掌握决策树的分类原理、实质和过程，并会采用决策树模型根据给定的数据进行实际应用，掌握决策树模型的编码实现。

### 3. 实验内容

1、分析空气质量监测数据，利用分类树对空气质量等级进行多分类预测。

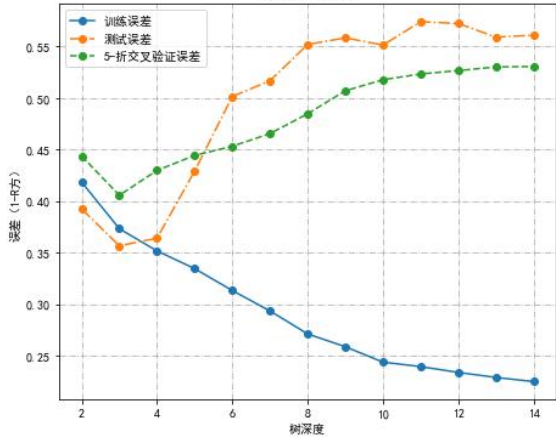
2、不同纯度度量方法（信息熵、基尼系数）的函数图像绘制。

二分类下的基尼系数和熵



3、树的深度对预测准确率的影响。

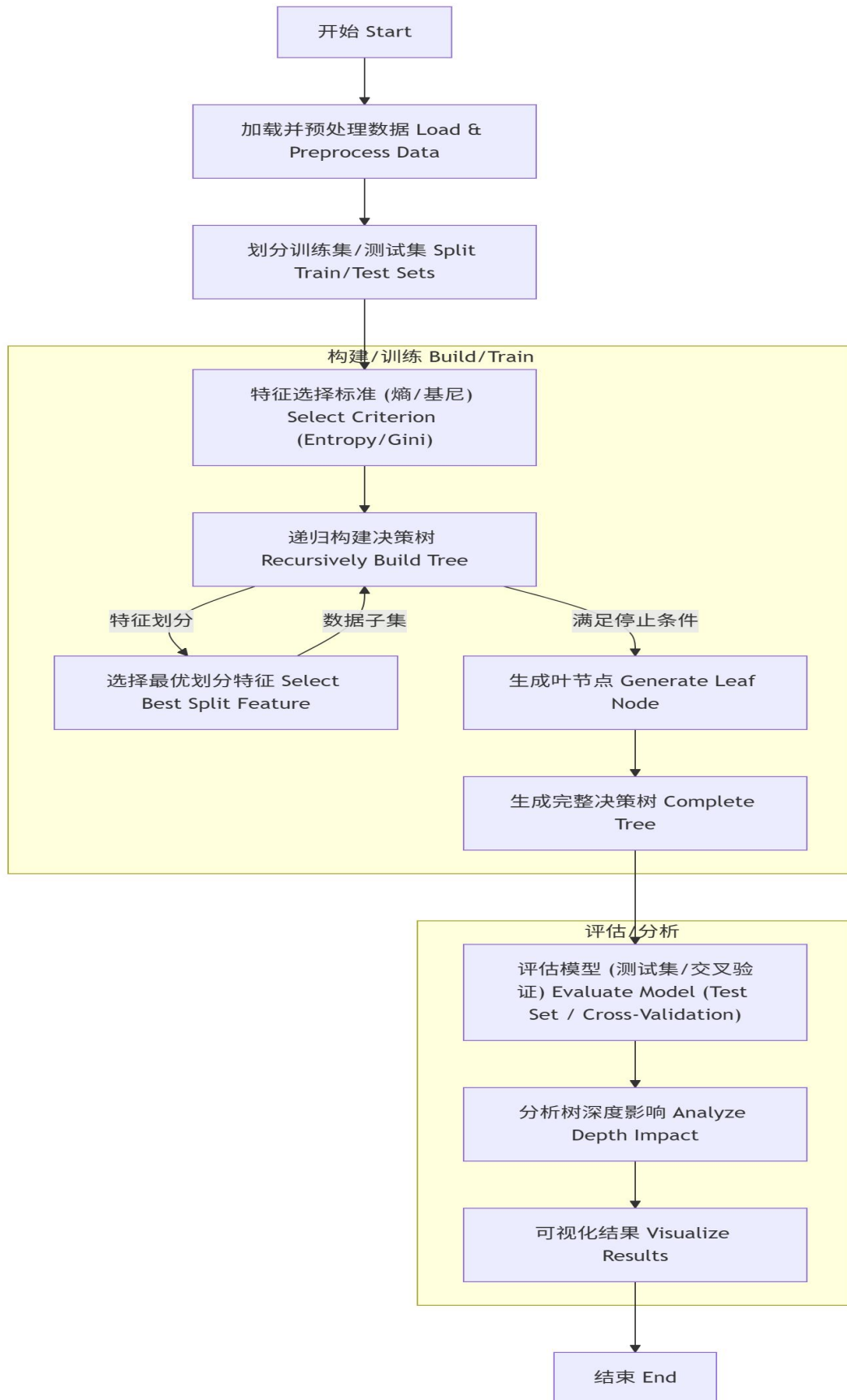
树深度和误差



### 4、 决策树的剪枝

4、 决策树的剪枝

#### 4. 实验原理或流程图



## 5. 实验过程或源代码

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split,
cross_val_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score,
classification_report, confusion_matrix
from sklearn.preprocessing import LabelEncoder
import graphviz # Optional: for visualizing the tree structure
from sklearn.tree import export_graphviz
import pydotplus # Optional: dependency for graphviz display
import os

# --- 1. 配置环境 (Environment Setup) ---

# 解决 Matplotlib 中文显示问题

plt.rcParams['font.sans-serif'] = ['SimHei'] # 'SimHei' 是黑
体

plt.rcParams['axes.unicode_minus'] = False # 解决坐标轴负
号显示问题

# --- 2. 加载数据 (Load Data) ---
file_path =
'D:\PaddlePaddle-EfficientNetV2\PaddleClas-EfficientNet\北 京
市空气质量数据.xlsx' # <--- 修改为你实际的文件路径
try:
    df = pd.read_excel(file_path)
    print("数据加载成功:")
    print(df.head())
    print("\n 数据信息:")
    df.info()
```

```
except FileNotFoundError:
    print(f"错误：找不到文件 '{file_path}'。请确保文件路径正确。")
    exit()
except Exception as e:
    print(f"加载数据时出错：{e}")
    exit()

# --- 3. 数据预处理 (Data Preprocessing) ---

# 假设最后一列是目标变量 'AQI 等级'，其余为特征
# target_column = df.columns[-1] # <--- 这是错误假设！
# feature_columns = df.columns[:-1].tolist() # <--- 这包含了不合适的列

target_column = '质量等级' # <--- 修改：明确指定目标列
# 选择合适的特征列，排除非数值列、目标列和可能冗余的列（如 AQI 本身）
feature_columns = ['PM2.5', 'PM10', 'SO2', 'CO', 'NO2', 'O3'] # <--- 修改：选择实际用于预测的特征

X = df[feature_columns]
y_raw = df[target_column]

# 将分类目标变量编码为数字
le = LabelEncoder()
y = le.fit_transform(y_raw)
class_names = le.classes_ # 保存原始类别名称，方便后续解释

print(f"\n 特征列：{feature_columns}")
```

```

print(f"目标列: {target_column}")

print(f"    编    码    后    的    目    标    类    别    映    射    :
{dict(zip(le.transform(class_names), class_names))}")
print(f"原始类别分布:\n{y_raw.value_counts()}")

# --- 4. 划分训练集和测试集 (Split Data) ---
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y # stratify
    保证类别比例
)
print(f"\n 训练集大小: {X_train.shape}, 测试集大小:
{X_test.shape}")

# --- 5. 任务二: 绘制不同纯度度量函数图像 (Plot Impurity
Measures) ---
def entropy(p):
    """计算二分类信息熵"""

    p = np.clip(p, 1e-10, 1 - 1e-10) # 避免 log(0)
    return -p * np.log2(p) - (1 - p) * np.log2(1 - p)

def gini_index(p):
    """计算二分类基尼系数"""

    return 2 * p * (1 - p) # 等价于 1 - (p**2 + (1-p)**2)

p_values = np.linspace(0.001, 0.999, 200)
entropy_values = entropy(p_values)
gini_values = gini_index(p_values)

plt.figure(figsize=(8, 6))
plt.plot(p_values, entropy_values, label='熵 (Entropy)',
linestyle='-')

```

```

plt.plot(p_values, gini_values, label='基尼系数 (Gini)',
linestyle='-.') # 使用与示例图一致的线型

plt.title('二分类下的基尼系数和熵 (Gini Index and Entropy
for Binary Classification)')
plt.xlabel('概率 P (Probability P)')

plt.ylabel('计算结果 (Calculated Value)')
plt.legend()
plt.grid(True, linestyle='-.')
plt.ylim(bottom=0) # 确保 Y 轴从 0 开始
plt.show()

# --- 6. 任务一 & 三：训练决策树并分析树深度影响 (Train
Tree & Analyze Depth) ---
max_depths = range(1, 15) # 测试的树深度范围
train_errors = []
test_errors = []
cv_errors = [] # 5 折交叉验证误差

print("\n 开始分析树深度对误差的影响...")
for depth in max_depths:
    # 使用信息熵作为标准 (也可以改为 'gini')
    dt_clf = DecisionTreeClassifier(criterion='entropy',
max_depth=depth, random_state=42)

    # 1. 训练误差
    dt_clf.fit(X_train, y_train)
    y_train_pred = dt_clf.predict(X_train)
    train_accuracy = accuracy_score(y_train, y_train_pred)
    train_errors.append(1 - train_accuracy)

    # 2. 测试误差

```

```

y_test_pred = dt_clf.predict(X_test)
test_accuracy = accuracy_score(y_test, y_test_pred)
test_errors.append(1 - test_accuracy)

# 3. 5 折交叉验证误差 (在训练集上进行)

# 注意：交叉验证评估的是模型在训练数据上的泛化能力，避免了对单一测试集划分的依赖

cv_scores = cross_val_score(dt_clf, X_train, y_train, cv=5, scoring='accuracy')
mean_cv_accuracy = np.mean(cv_scores)
cv_errors.append(1 - mean_cv_accuracy)

print(f"          深度 = {depth}:    训练误差 = {1-train_accuracy:.4f}, 测试误差={1-test_accuracy:.4f}, 5 折 CV 误差={1-mean_cv_accuracy:.4f}")

# 绘制树深度与误差的关系图
plt.figure(figsize=(10, 7))
plt.plot(max_depths, train_errors, marker='o', linestyle='-', label='训练误差 (Training Error)')
plt.plot(max_depths, test_errors, marker='o', linestyle='-.', label='测试误差 (Test Error)')
plt.plot(max_depths, cv_errors, marker='o', linestyle='--', label='5-折交叉验证误差 (5-Fold CV Error)')

plt.title('树深度和误差 (Tree Depth vs. Error)')

plt.xlabel('树深度 (Tree Depth)')

# 保持与示例图一致的 Y 轴标签，即使 R 方 用于回归
plt.ylabel('误差 (1-准确率) Error (1-Accuracy)')

plt.xticks(max_depths)
plt.legend()

```



```
plt.grid(True, linestyle='-.')
plt.show()

# --- 7. 选择最优深度并评估最终模型 (Select Optimal Depth
& Evaluate Final Model) ---
# 根据上图选择一个较优的深度，通常是测试误差或交叉验证
误差开始稳定或上升的拐点前的深度

# 例如，可以找交叉验证误差最小的深度
optimal_depth_cv = max_depths[np.argmin(cv_errors)]
optimal_depth_test = max_depths[np.argmin(test_errors)] # 也
可以基于测试误差

# 在实践中，通常基于交叉验证选择超参数
optimal_depth = optimal_depth_cv
print(f"\n 根据 5 折交叉验证，建议的最优深度约为：
{optimal_depth}")
print(f"( 测试误差最小时对应的深度为：
{optimal_depth_test})")

# 使用选定的最优深度重新训练模型（可以在整个训练集上
训练）
final_clf = DecisionTreeClassifier(criterion='entropy',
max_depth=optimal_depth, random_state=42)
final_clf.fit(X_train, y_train)

# 在测试集上进行最终评估
y_final_pred = final_clf.predict(X_test)
final_accuracy = accuracy_score(y_test, y_final_pred)
print(f"\n 使用最优深度 {optimal_depth} 在测试集上的最终
评估:")
```

```
print(f"    准确率: {final_accuracy:.4f}")

print("    分类报告:")

# target_names 使用原始类别名称
print(classification_report(y_test, y_final_pred,
target_names=class_names))

print("    混淆矩阵:")

print(confusion_matrix(y_test, y_final_pred))

# 如果需要显示原始类别标签的混淆矩阵
conf_matrix = confusion_matrix(y_test, y_final_pred)
conf_df = pd.DataFrame(conf_matrix, index=class_names,
columns=class_names)

print("\n 混淆矩阵 (带标签):")
print(conf_df)


# --- 8. 任务四：决策树剪枝 (Pruning) ---

# 上述通过选择 'max_depth' 实际上就是一种预剪枝
(Pre-pruning) 策略。

# Scikit-learn 也支持基于代价复杂度剪枝 (Cost-Complexity
Pruning, 'ccp_alpha') 的后剪枝，

# 但对于本实验，通过分析深度影响来选择最优深度已满足要
求。

print(f"\n 剪枝说明：通过限制最大深度
(max_depth={optimal_depth}) 进行了预剪枝，以防止模型过拟
合。")
```

```

# --- 9. (可选) 可视化决策树 (Optional: Visualize Tree) ---

# 需要安装 graphviz (软件本身) 和 pydotplus (Python 库)

# 在命令行运行: pip install pydotplus graphviz

# 可能还需要将 graphviz 的 bin 目录添加到系统 PATH
try:
    dot_data = export_graphviz(final_clf,
                                out_file=None,

    feature_names=feature_columns,

    class_names=class_names, # 使用原始类别名称

                                filled=True,
                                rounded=True,

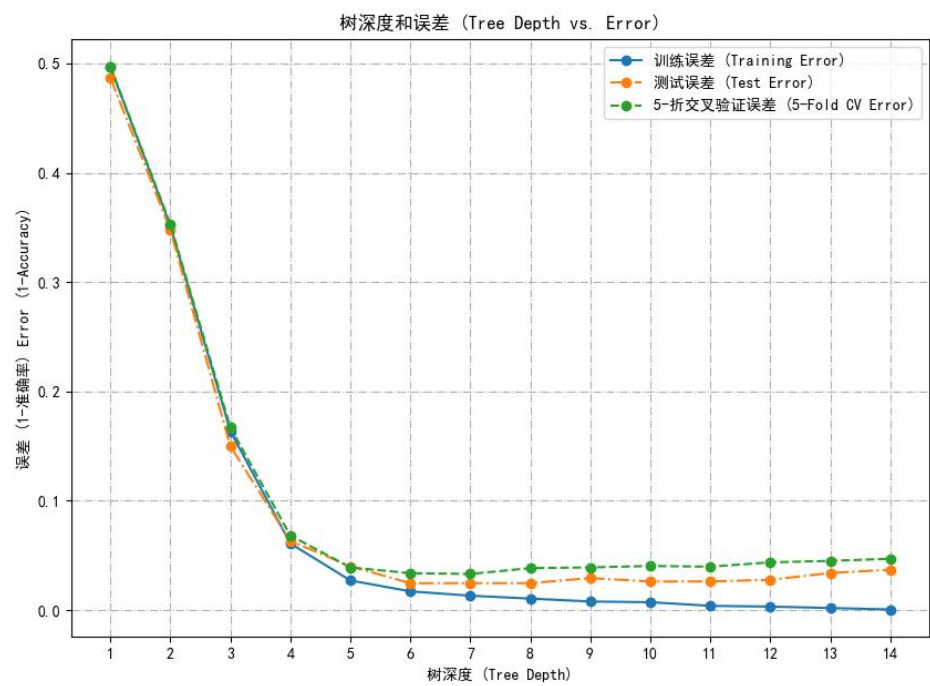
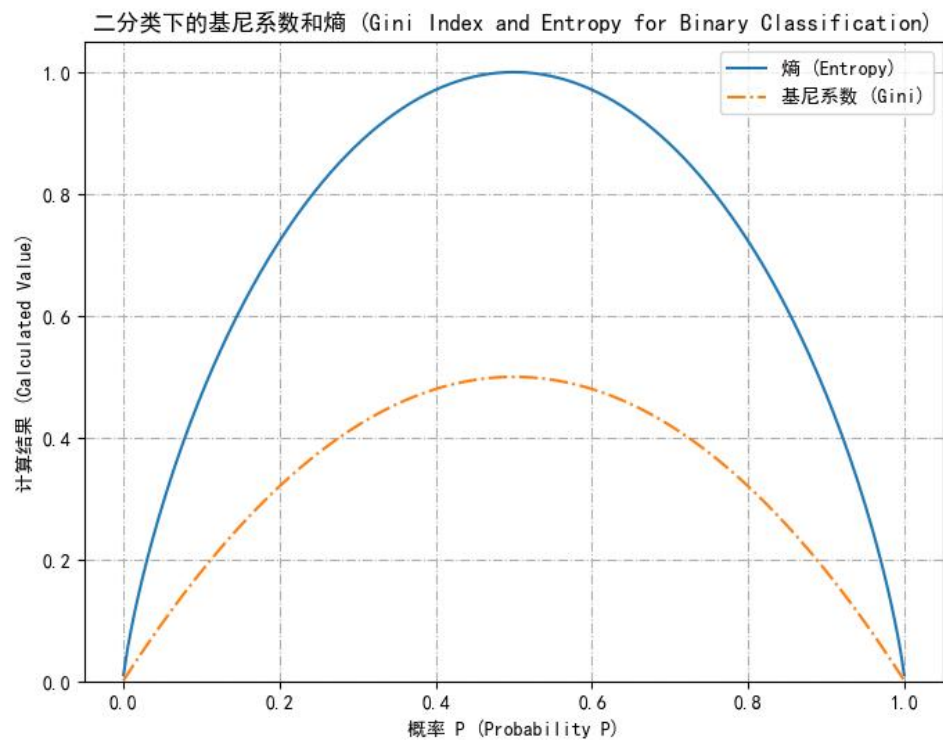
    special_characters=True)
    graph = pydotplus.graph_from_dot_data(dot_data)
    # 保存为图片文件
    tree_image_path = 'air_quality_decision_tree.png'
    graph.write_png(tree_image_path)
    print(f"\n 决策树结构已保存为图片: {tree_image_path}")

    # 如果在 Jupyter Notebook 中, 可以直接显示
    # from IPython.display import Image
    # display(Image(graph.create_png()))
except ImportError:
    print("\n 无法导入 pydotplus 或 graphviz 未正确配置,
    跳过决策树可视化。")

    print(" 请安装 graphviz 软件并运行 'pip install
    pydotplus'")
except Exception as e:
    print(f"\n 生成决策树可视化时出错: {e}")

```

```
print("\n 实验完成!")
```



```
C:\Users\86198\.conda\envs\Y0L0v8\python.exe D:\PaddlePaddle-EfficientNetV2\PaddleClas-EfficientNet\test_3.py
数据加载成功:
      日期    AQI  质量等级  PM2.5  PM10  SO2   CO   NO2   O3
0 2014-01-01    81      良      45   111   28  1.5   62   52
1 2014-01-02   145  轻度污染    111   168   69  3.4   93   14
2 2014-01-03    74      良      47    98   29  1.3   52   56
3 2014-01-04   149  轻度污染    114   147   40  2.8   75   14
4 2014-01-05   119  轻度污染     91   117   36  2.3   67   44

数据信息:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2155 entries, 0 to 2154
Data columns (total 9 columns):
 #   Column  Non-Null Count  Dtype
---  ---
0   日期      2155 non-null  datetime64[ns]
1   AQI       2155 non-null  int64
2   质量等级  2155 non-null  object
3   PM2.5     2155 non-null  int64
4   PM10      2155 non-null  int64
5   SO2       2155 non-null  int64
6   CO        2155 non-null  float64
7   NO2       2155 non-null  int64
8   O3        2155 non-null  int64
dtypes: datetime64[ns](1), float64(1), int64(6), object(1)
memory usage: 151.6+ KB
```

```
特征列: ['PM2.5', 'PM10', 'SO2', 'CO', 'NO2', 'O3']
目标列: 质量等级
编码后的目标类别映射: {0: '严重污染', 1: '中度污染', 2: '优', 3: '无', 4: '良', 5: '轻度污染', 6: '重度污染'}
原始类别分布:
质量等级
良      842
轻度污染  487
优      383
中度污染  261
重度污染  130
严重污染   46
无         6
Name: count, dtype: int64

训练集大小: (1508, 6), 测试集大小: (647, 6)
```

深度=14：训练误差=0.0007，测试误差=0.0371，5折CV误差=0.0471

根据5折交叉验证，建议的最优深度约为：7

(测试误差最小时对应的深度为：6)

使用最优深度 7 在测试集上的最终评估：

准确率：0.9753

分类报告：

	precision	recall	f1-score	support
严重污染	1.00	0.93	0.96	14
中度污染	0.99	0.95	0.97	78
优	0.98	0.97	0.97	115
无	1.00	0.50	0.67	2
良	0.97	1.00	0.98	253
轻度污染	0.98	0.97	0.98	146
重度污染	0.95	0.97	0.96	39
accuracy			0.98	647
macro avg	0.98	0.90	0.93	647
weighted avg	0.98	0.98	0.97	647

混淆矩阵:

```
[[ 13  0  0  0  0  0  1]
 [  0 74  1  0  0  2  1]
 [  0  0 111  0  4  0  0]
 [  0  0  0  1  1  0  0]
 [  0  0  0  0 252  1  0]
 [  0  1  1  0  2 142  0]
 [  0  0  0  0  1  0 38]]
```

混淆矩阵 (带标签):

	严重污染	中度污染	优	无	良	轻度污染	重度污染
严重污染	13	0	0	0	0	0	1
中度污染	0	74	1	0	0	2	1
优	0	0	111	0	4	0	0
无	0	0	0	1	1	0	0
良	0	0	0	0	252	1	0
轻度污染	0	1	1	0	2	142	0
重度污染	0	0	0	0	1	0	38

剪枝说明: 通过限制最大深度 (max\_depth=7) 进行了预剪枝, 以防止模型过拟合。

生成决策树可视化时出错: GraphViz's executables not found

实验完成!

## 6. 实验心得

数据加载和预处理成功:

现在程序正确地将 质量等级 作为目标列, 并将 PM2.5 到 O3 的 6 个指标作为特征列。

LabelEncoder 也正确地将文本的质量等级 (严重污染、中度污染、优、无、良、轻度污染、重度污染) 映射成了数字 0 到 6。

原始类别分布 显示了各个等级的样本数量, 可以看到 "良" 最多, "无" 最少 (只有 6 个样本)。

训练/测试集划分：数据成功划分为训练集（1508 条）和测试集（647 条）。  
交叉验证警告（UserWarning）：

程序在进行 5 折交叉验证(cross\_val\_score)时，反复出现警告：The least populated class in y has only 4 members, which is less than n\_splits=5.

原因：这是因为训练集（1508 条）中，“无”这个类别只有大约  $1508 / 2155 * 6 \approx 4$  个样本。进行 5 折交叉验证时，需要将训练集分成 5 份。由于“无”类别的样本太少（只有 4 个），无法保证每一折（份）都至少包含一个“无”类别的样本来进行分层抽样。

影响：这只是一个警告，scikit-learn 仍然会尝试完成交叉验证，但关于“无”这个类别的交叉验证评估可能不太稳定或准确。不过，对于评估模型整体性能（尤其是占多数的类别）通常影响不大。

树深度与误差分析：

训练误差：随着树深度的增加，训练误差持续降低，从深度 1 的约 0.497 下降到深度 14 的接近 0.0007。这符合预期，因为更深的树能更精细地拟合训练数据。

测试误差 & 5 折 CV 误差：这两个误差一开始随着深度增加而显著下降（模型学习到了有效模式），在深度大约 6 或 7 附近达到最小值（测试误差约 0.0247，CV 误差约 0.0332），之后开始缓慢上升或波动。

过拟合迹象：当深度超过 7 左右，训练误差继续下降，但测试/CV 误差不再下降甚至开始上升，这表明模型开始过拟合训练数据中的噪声或特定模式，导致在新数据（测试集/CV 验证集）上的泛化能力下降。

最优深度选择：

根据 5 折交叉验证的误差，脚本建议的最优深度是 7（CV 误差约 0.0332）。

根据测试集误差，最优深度是 6（测试误差约 0.0247）。

两者非常接近，选择 7 作为最优深度是合理的（通常优先考虑交叉验证结果，因为它更稳健）。

最终模型评估（最优深度=7）：

准确率：在测试集上达到了惊人的 0.9753（97.53%），说明模型预测空气质量等级的效果非常好。

分类报告：

大部分类别的 precision（精确率）、recall（召回率）和 f1-score 都很高（接近或超过 0.95）。

对于样本极少的“无”类别（测试集中只有 2 个样本），召回率是 0.50（找到了其中的 1 个），精确率是 1.00（预测为“无”的那个确实是“无”），F1 分数是 0.67。这个类别的指标因为样本太少，参考价值有限。

macro avg（宏平均，不考虑样本量）的召回率（0.90）被“无”拉低了，但精确率和 F1 仍然很高。

weighted avg（加权平均，考虑样本量）的各项指标都非常高（约 0.97-0.98），更能反映模型在整体数据上的表现。

混淆矩阵：对角线上的数字（正确预测数）远大于非对角线上的数字（错误预测数），直观显示了模型的高性能。可以看到少量样本被分错到了邻近的等级，这在实际中是常见的。

剪枝说明：正确指出通过限制 max\_depth=7 进行了预剪枝。

可视化错误：GraphViz's executables not found 这个错误表明你的电脑



上没有安装 Graphviz 软件,或者它的可执行文件路径没有添加到系统环境变量 PATH 中。这导致 pydotplus 无法调用 Graphviz 来生成决策树图片。这不影响模型训练和评估的结果,只是无法生成那张树状图。

总结:

代码修改成功解决了之前的错误。实验运行顺利,决策树模型在空气质量数据集上表现出了非常高的分类准确率。深度分析显示,选择一个适中的深度(如 7)可以有效防止过拟合,获得良好的泛化性能。关于样本极少的 "无" 类别的警告和评估指标需要注意,但整体模型效果优异。可视化失败是环境配置问题,不影响核心实验结论。