姓名：曾闻天　学号：231880147　2025 年 11 月 10 日

一. (30 points) 线性回归

给定包含 m 个样例的数据集 $D = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \cdots, (\mathbf{x}_m, y_m)\}$，其中 $\mathbf{x}_i = (x_{i1}; x_{i2}; \cdots; x_{id}) \in \mathbb{R}^d$, $y_i \in \mathbb{R}$ 为 $\mathbf{x}_i$ 的实数标记。线性回归模型要求该线性模型的预测结果和其对应的标记之间的误差平方之和最小:

$$(\mathbf{w}^*, b^*) = \underset{(\mathbf{w},b)}{\operatorname{argmin}} \frac{1}{2} \sum_{i=1}^{m} (y_i - (\mathbf{w}^\top \mathbf{x}_i + b))^2 \tag{1}$$

即寻找一组权重 $(\mathbf{w}, b)$，使其对 D 中示例预测的整体误差最小。定义 $y = [y_1; y_2; \cdots; y_m] \in \mathbb{R}^m$ 且 $\mathbf{X} = [\mathbf{x}_1^\top; \mathbf{x}_2^\top; \cdots; \mathbf{x}_m^\top] \in \mathbb{R}^{m \times d}$.

1. 请将线性回归的优化过程使用矩阵进行表示; (10 points)

2. 使用矩阵形式给出权重 $\mathbf{w}^*$ 和偏移 b^* 最优解的表达式；(10 points)

3. 针对波士顿房价预测数据集 (boston), 编程实现原始线性回归模型。请基于闭式解在训练集上构建模型, 计算测试集上的均方误差 (Mean Square Error, MSE)。请参考如下形式完成函数 linear_regression 和 MSE 的代码。除示例代码中使用到的 sklearn 库函数外, 不能使用其他的 sklearn 函数, 需要基于 numpy 实现线性回归模型闭式解以及 MSE 的计算. (10 points)

In [3]:
```python
from sklearn.datasets import load_boston
from sklearn.model_selection import train_test_split

X, y = load_boston(return_X_y=True)
trainx , testx , trainy , testy = train_test_split(X, y, test_size = 0.33, random_state =42)

def linear_regression(X_train , y_train):
''' 线性回归
    : 参数X_train: np.ndarray , 形状为(n, d), n 个d 维训练样本
    : 参数y_train: np.ndarray , 形状为(n, 1), 每个样本的标签
    : 返回: 权重矩阵W
'''

def MSE(X_train , y_train , X_test , y_test):
''' 调用linear_regression得到权重矩阵W后计算MSE
    : 参数X_train: np.ndarray , 形状为(n, d), n 个d 维训练样本
    : 参数y_train: np.ndarray , 形状为(n, 1), 每个训练样本的标签
    : 参数X_test: np.ndarray , 形状为(m, d), m 个d 维测试样本
    : 参数y_test: np.ndarray , 形状为(m, 1), 每个测试样本的标签
    : 返回: 标量, MSE 值
'''


linear_regression_MSE= MSE(trainx , trainy , testx , testy)
```

图 1: 示例代码

解:

1.

将线性模型写为矩阵形式。令

$$\tilde{X} = [X, 1_m] \in \mathbb{R}^{m\times(d+1)}, \quad 1_m = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}_{m\times 1}$$

参数向量

$$\theta = \begin{bmatrix} w \\ b \end{bmatrix} \in \mathbb{R}^{d+1}$$

则预测值为 $\tilde{X}\theta$，残差向量为 $y - \tilde{X}\theta$。

优化目标可表示为：

$$\min_{\theta\in\mathbb{R}^{d+1}} \frac{1}{2} \left\| y - \tilde{X}\theta \right\|_2^2$$

2.

由最小二乘理论，最优解满足正规方程：

$$\tilde{X}^\top \left(y - \tilde{X}\theta \right) = 0$$

即

$$\tilde{X}^\top \tilde{X}\theta = \tilde{X}^\top y$$

若 $\tilde{X}^\top \tilde{X}$ 可逆，则最优解为：

$$\theta^* = \left(\tilde{X}^\top \tilde{X} \right)^{-1} \tilde{X}^\top y$$

展开为：

$$\begin{bmatrix} w^* \\ b^* \end{bmatrix} = \left([X, 1_m]^\top [X, 1_m] \right)^{-1} [X, 1_m]^\top y$$

3.

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

```

```python
def linear_regression(X_train, y_train):
    ''' 线性回归
    : 参数X_train: np.ndarray , 形状为(n, d), n 个d 维训练样本
    : 参数y_train: np.ndarray , 形状为(n, 1), 每个样本的标签
    : 返回: 权重矩阵W
    '''
    X_train_with_bias = np.column_stack([X_train, np.ones(X_train.shape[0])])
    W = np.linalg.pinv(X_train_with_bias.T @ X_train_with_bias) @ X_train_with_bias.T @ y_train
    return W

def MSE(X_train, y_train, X_test, y_test):
    ''' 调用linear_regression得到权重矩阵W后计算MSE
    : 参数X_train: np.ndarray , 形状为(n, d), n 个d 维训练样本
    : 参数y_train: np.ndarray , 形状为(n, 1), 每个训练样本的标签
    : 参数X_test: np.ndarray , 形状为(m, d), m 个d 维测试样本
    : 参数y_test: np.ndarray , 形状为(m, 1), 每个测试样本的标签
    : 返回: 标量, MSE 值
    '''
    W = linear_regression(X_train, y_train)
    X_test_with_bias = np.column_stack([X_test, np.ones(X_test.shape[0])])
    y_pred = X_test_with_bias @ W
    mse = np.mean((y_test - y_pred) ** 2)
    return mse

# 从本地CSV文件加载波士顿数据集
file_path = r"C:\Users\Administrator\Downloads\boston.csv"
try:
    boston_data = pd.read_csv(file_path)
    X = boston_data.iloc[:, :-1].values
    y = boston_data.iloc[:, -1].values
except FileNotFoundError:
    exit()
except Exception as e:
    exit()

trainx, testx, trainy, testy = train_test_split(X, y, test_size=0.33, random_state=42)

linear_regression_MSE = MSE(trainx, trainy, testx, testy)
print(f"线性回归在测试集上的MSE: {linear_regression_MSE:.4f}")
```

```
问题  输出  调试控制台  终端  端口

PS C:\Users\Administrator\Downloads> python -u "c:\Users\Administrator\Downloads\1.py"
线性回归在测试集上的MSE: 20.7240
PS C:\Users\Administrator\Downloads>
```

图 2: 运行结果

根据运行结果，MSE = 20.72

二. (25 points) 对率回归

信用卡欺诈检测数据集 (Credit Card Fraud Detection) 包含了 2013 年 9 月通过信用卡进行的欧洲持卡人的交易。这是一个典型的类别不平衡数据集，数据集中正常交易的标签远多于欺诈交易。请你根据附件中提供的该数据集完成以下问题:

1. 数据集共有 284807 个样本，其中只有 492 个负样本。请按照训练集和测试集比例 7:3 的方式划分数据集（使用固定的随机种子）。在训练集上分别训练对率回归 (Logistic Regression) 与决策树两类模型，并计算两者在测试集上的精度（至少包含：准确率、召回率、F1 分数、AUC）。请展示完整代码，并在同一张图上绘制两种模型的 ROC 曲线。(5 points)

2. 保持测试集不变，在训练集中对多数类（正例）进行随机下采样，分别构造 **负例: 正例**的三种比例：**1 : 10**、**1 : 100**、**1 : 200**。在这三个新的训练集上分别训练对率回归与决策树模型，并记录每个模型的精度。观察并比较这几组实验的结果，结合准确率与召回率的定义，请说明不平衡数据集对模型的影响（8 points）;

3. 除了上述第 2 问的随机欠采样的方式以外，对小类样本的“过采样”也是处理不平衡问题的基本策略。一种经典的方法为人工合成的过采样技术 (Synthetic Minority Over-sampling Technique, SMOTE), 其在合成样本时寻找小类中某一个样本的近邻, 并在该样本与近邻之间进行差值, 作为合成的新样本。请查阅相关资料，实现 SMOTE 算法中的 over sampling 函数, 以代码块的形式附于下方即可（8 points）;

```python
"""
注意:
1. 这个框架提供了基本的结构，您需要完成所有标记为 'pass' 的函数。
2. 记得处理数值稳定性问题，例如在计算对数时避免除以零。
3. 在报告中详细讨论您的观察结果和任何有趣的发现。
"""
class SMOTE(object):
    def __init__(self , X, y, N, K, random_state =0):
        self.N = N # 每个小类样本合成样本个数
        self.K = K # 近邻个数
        self.label = y # 进行数据增强的类别
        self.sample = X
```

```
        self.n_sample , self.n = self.sample.shape # 获得样本个数，特征个数

    def over_sampling(self):
        pass
```

Listing 1: SMOTE 模型接口

4. 请说明 SMOTE 算法的缺点并讨论可能的改进方案（4 points）。

解:

1.

```
import pandas as pd
import numpy as np
import matplotlib
matplotlib.use('TkAgg')
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, recall_score, f1_score, roc_auc_score,
    roc_curve
import warnings
warnings.filterwarnings('ignore')

plt.rcParams['font.sans-serif'] = ['SimHei', 'Microsoft YaHei', 'DejaVu Sans']
plt.rcParams['axes.unicode_minus'] = False

np.random.seed(42)

try:
    df = pd.read_csv('creditcard.csv')
except FileNotFoundError:
    exit()

X = df.drop('Class', axis=1)
y = df['Class']

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)

log_reg = LogisticRegression(random_state=42, max_iter=1000, class_weight='balanced')
decision_tree = DecisionTreeClassifier(random_state=42, class_weight='balanced')
```

```
33
34 log_reg.fit(X_train, y_train)
35 decision_tree.fit(X_train, y_train)
36
37 y_pred_log_reg = log_reg.predict(X_test)
38 y_pred_tree = decision_tree.predict(X_test)
39
40 y_pred_proba_log_reg = log_reg.predict_proba(X_test)[:, 1]
41 y_pred_proba_tree = decision_tree.predict_proba(X_test)[:, 1]
42
43 def evaluate_model(y_true, y_pred, y_pred_proba, model_name):
44     accuracy = accuracy_score(y_true, y_pred)
45     recall = recall_score(y_true, y_pred)
46     f1 = f1_score(y_true, y_pred)
47     auc = roc_auc_score(y_true, y_pred_proba)
48     print(f"\n{model_name} 模型评估:")
49     print(f"准确率 (Accuracy): {accuracy:.4f}")
50     print(f"召回率 (Recall): {recall:.4f}")
51     print(f"F1分数: {f1:.4f}")
52     print(f"AUC: {auc:.4f}")
53     return accuracy, recall, f1, auc
54
55 print("\n" + "="*50)
56 acc_lr, recall_lr, f1_lr, auc_lr = evaluate_model(y_test, y_pred_log_reg,
       y_pred_proba_log_reg, "逻辑回归")
57
58 print("\n" + "="*50)
59 acc_dt, recall_dt, f1_dt, auc_dt = evaluate_model(y_test, y_pred_tree, y_pred_proba_tree,
       "决策树")
60
61 plt.figure(figsize=(10, 8))
62 fpr_lr, tpr_lr, _ = roc_curve(y_test, y_pred_proba_log_reg)
63 fpr_dt, tpr_dt, _ = roc_curve(y_test, y_pred_proba_tree)
64 plt.plot(fpr_lr, tpr_lr, label=f'逻辑回归 (AUC = {auc_lr:.4f})', linewidth=2, color='blue'
       )
65 plt.plot(fpr_dt, tpr_dt, label=f'决策树 (AUC = {auc_dt:.4f})', linewidth=2, color='red')
66 plt.plot([0, 1], [0, 1], 'k--', label='随机分类器', alpha=0.5)
67 plt.xlim([0.0, 1.0])
68 plt.ylim([0.0, 1.05])
69 plt.xlabel('假正率 (False Positive Rate)', fontsize=12)
70 plt.ylabel('真正率 (True Positive Rate)', fontsize=12)
71 plt.title('ROC曲线比较 - 信用卡欺诈检测', fontsize=14, fontweight='bold')
72 plt.legend(loc='lower right', fontsize=12)
73 plt.grid(True, alpha=0.3)
74 plt.savefig('roc_curve_comparison.png', dpi=300, bbox_inches='tight')
75 plt.tight_layout()
76 plt.show()
```

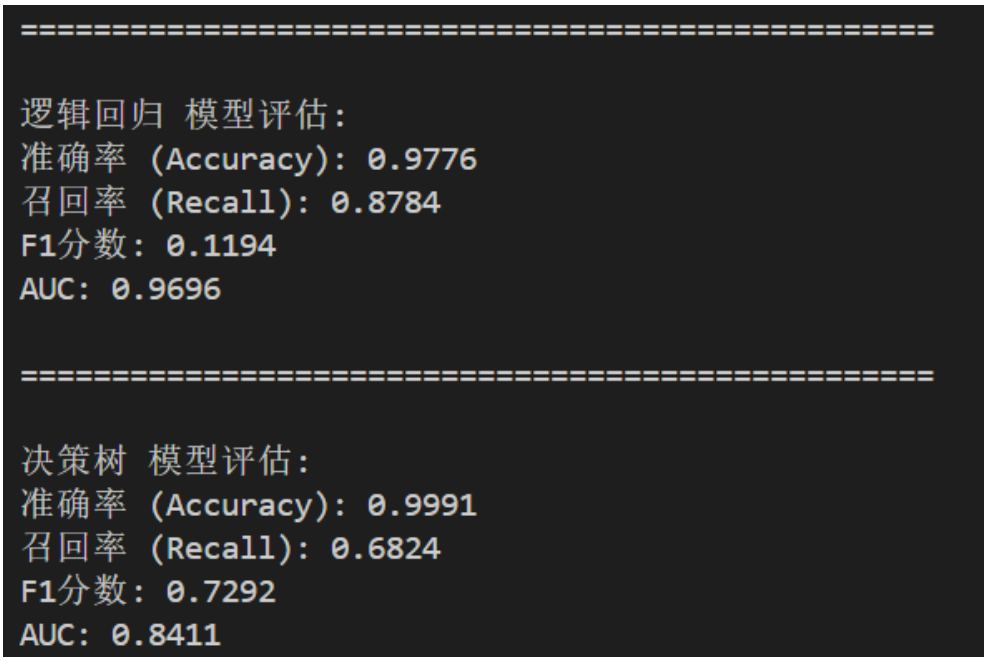

图 3: 运行结果

可知:

逻辑回归模型评估: 准确率 (Accuracy): 0.9776; 召回率 (Recall): 0.8784; F1 分数: 0.1194; AUC: 0.9696;

决策树模型评估: 准确率 (Accuracy): 0.9991; 召回率 (Recall): 0.6824; F1 分数: 0.7292; AUC: 0.8411.

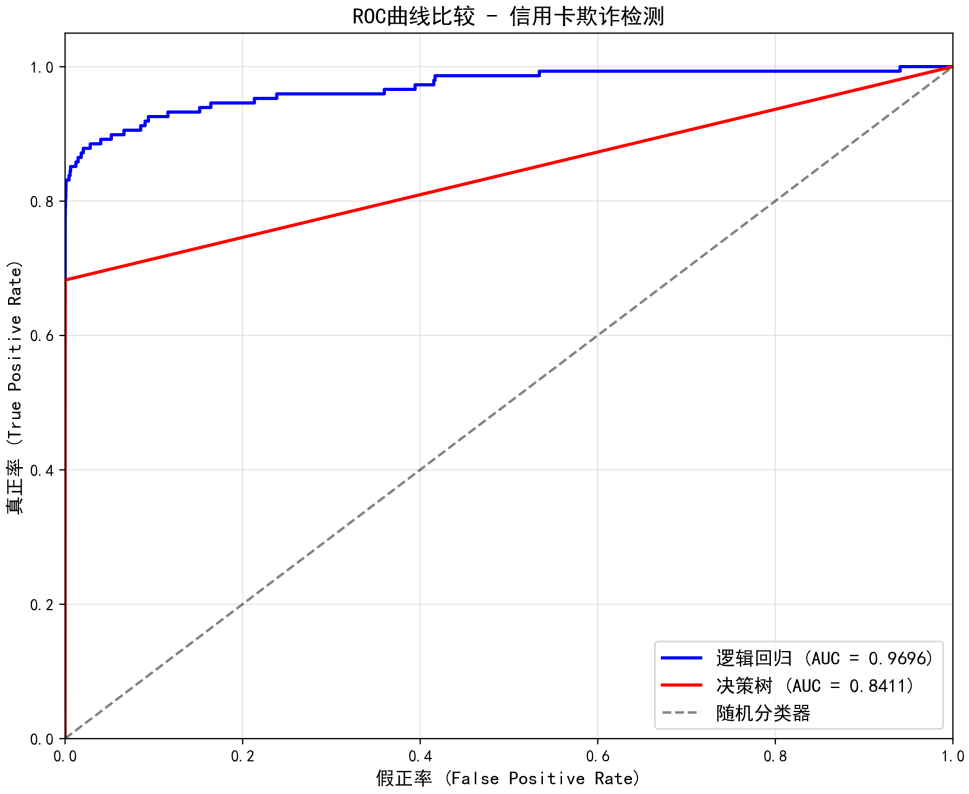

图 4: ROC 曲线

2.

1. 不同比例的影响:
1:10 比例: 正负样本相对平衡，模型能较好识别两类样本
1:100 比例: 开始偏向多数类，但仍有足够信息识别少数类
1:200 比例: 高度不平衡，模型可能难以学习少数类的特征

2. 准确率, 召回率的权衡:
在不平衡数据集中，如果模型总是预测多数类，准确率会很高但召回率为 0
下采样后，模型更关注少数类，召回率提高但准确率可能下降
需要根据业务需求在准确率和召回率之间找到平衡

3. 不平衡数据集对模型的主要影响:
评估指标误导：准确率不能真实反映模型对少数类的识别能力
性能权衡：需要在召回率（不漏检）和精确率（减少误报）之间找到平衡
模型偏见：模型倾向于学习多数类的模式，忽视少数类特征
业务影响：需要根据具体业务需求选择合适的不平衡处理策略

表 1: 实验结果总结

数据比例	模型	准确率	精确率	召回率	F1 分数	AUC
1:10	逻辑回归	0.9974	0.3844	0.8311	0.5256	0.9684
	决策树	0.9842	0.0842	0.8243	0.1528	0.9044
1:100	逻辑回归	0.9992	0.7682	0.7838	0.7759	0.9659
	决策树	0.9977	0.4113	0.7838	0.5395	0.8909
1:200	逻辑回归	0.9992	0.7926	0.7230	0.7562	0.9609
	决策树	0.9983	0.5160	0.7635	0.6158	0.8811

3.

```python
class SMOTE(object):
    def __init__(self, X, y, N, K, random_state=0):
        self.N = N  # 每个小类样本合成样本个数
        self.K = K  # 近邻个数
        self.label = y  # 进行数据增强的类别
        self.sample = X
        self.n_sample, self.n = self.sample.shape  # 获得样本个数，特征个数
        self.random_state = random_state
        np.random.seed(random_state)

    def over_sampling(self):
        counter = Counter(self.label)
        minority_class = min(counter, key=counter.get)
        minority_indices = np.where(self.label == minority_class)[0]
        minority_samples = self.sample[minority_indices]

        n_minority = len(minority_samples)

        if self.K > n_minority - 1:
            self.K = n_minority - 1

        if self.N < 100:
            n_synthetic = n_minority * self.N
        else:
            n_majority = max(counter.values())
            n_synthetic = int(n_majority * (self.N / 100) - n_minority)

        knn = NearestNeighbors(n_neighbors=self.K + 1)
        knn.fit(minority_samples)
        distances, indices = knn.kneighbors(minority_samples)
        indices = indices[:, 1:]
```

```
        synthetic_samples = []

        for i in range(n_minority):
            current_sample = minority_samples[i]
            nn_indices = indices[i]
            if self.N < 100:
                n_generate = self.N
            else:
                n_generate = n_synthetic // n_minority
                if i < n_synthetic % n_minority:
                    n_generate += 1

            for j in range(n_generate):
                nn_index = np.random.choice(nn_indices)
                nn_sample = minority_samples[nn_index]

                gap = np.random.random()

                synthetic_sample = current_sample + gap * (nn_sample - current_sample)
                synthetic_samples.append(synthetic_sample)

        synthetic_samples = np.array(synthetic_samples)
        synthetic_labels = np.ones(len(synthetic_samples)) * minority_class

        X_resampled = np.vstack([self.sample, synthetic_samples])
        y_resampled = np.hstack([self.label, synthetic_labels])

        return X_resampled, y_resampled
```

4.

SMOTE 算法缺点：
生成样本可能过度拟合少数类边界。
在噪声附近生成样本会加剧噪声影响。
可能模糊类间边界，导致分类器性能下降。
在高维空间中，欧氏距离失效，生成样本质量差。
未考虑整体数据分布，可能生成不符合真实分布的样本。

改进方案：
根据局部密度或邻域信息调整合成策略（如 Borderline-SMOTE、ADASYN）。
在生成时加入约束条件，避免噪声区域合成（如 Safe-Level-SMOTE）。
结合集成学习，降低过拟合风险（如 SMOTE-Boost）。
利用流形结构生成更符合数据分布的样本（如 SMOTE-ENN）。

三. (20 points) 决策树

二分类数据集 D 含 20 个样本，正类 8、反类 12。候选离散属性为 A 与 B，其在 D 中的类分布如下，若需对数，取底数 2：

A 的取值	正类	反类	小计
$A{=}a_1$	2	8	10
$A{=}a_2$	6	4	10
合计	8	12	20

B 的取值	正类	反类	小计
$B{=}b_1$	3	2	5
$B{=}b_2$	1	4	5
$B{=}b_3$	4	6	10
合计	8	12	20

请完成：

1. 计算数据集 D 的信息熵 $\mathrm{Ent}(D)$ (5 points)；

2. 分别计算信息增益 $\mathrm{Gain}(D,A)$、$\mathrm{Gain}(D,B)$，以及对应的增益率 $\mathrm{GainRatio}(D,A)$、$\mathrm{GainRatio}(D,B)$ (5 points)；

3. 分别给出在 ID3（最大信息增益）、C4.5（最大增益率）、CART 分类树（最大基尼指数下降）三种准则下根节点最优划分的属性名称，并简述理由（可给出近似值）(10 points)。

解:

1.

$$\mathrm{Ent}(D) = -\sum_{k=1}^{|\mathcal{Y}|} p_k \log_2 p_k$$

其中 $|\mathcal{Y}| = 2$，$p_+ = \frac{8}{20} = 0.4$，$p_- = \frac{12}{20} = 0.6$。

$$\mathrm{Ent}(D) = -\left(0.4 \log_2 0.4 + 0.6 \log_2 0.6\right)$$

$$\mathrm{Ent}(D) \approx 0.970954$$

2.

对于 $A = a_1$：正类 2，反类 8，共 10 个样本。

$$p_+ = 0.2, \quad p_- = 0.8$$

$$\mathrm{Ent}(D^{a_1}) \approx 0.72193$$

对于 $A = a_2$：正类 6，反类 4，共 10 个样本。

$$p_+ = 0.6, \quad p_- = 0.4$$

$$\mathrm{Ent}(D^{a_2}) \approx 0.97095$$

条件熵：

$$\mathrm{Ent}_A(D) \approx 0.84644$$

信息增益：

$$\mathrm{Gain}(D, A) = \mathrm{Ent}(D) - \mathrm{Ent}_A(D) \approx 0.125$$

固有值（Intrinsic Value）：

$$\mathrm{IV}(A) = 1$$

增益率：

$$\mathrm{GainRatio}(D, A) = \frac{\mathrm{Gain}(D, A)}{\mathrm{IV}(A)} \approx 0.125$$

对于 $B = b_1$：正类 3，反类 2，共 5 个样本。

$$p_+ = 0.6, \quad p_- = 0.4$$

$$\mathrm{Ent}(D^{b_1}) \approx 0.97095$$

对于 $B = b_2$：正类 1，反类 4，共 5 个样本。

$$p_+ = 0.2, \quad p_- = 0.8$$

$$\mathrm{Ent}(D^{b_2}) \approx 0.72193$$

对于 $B = b_3$：正类 4，反类 6，共 10 个样本。

$$p_+ = 0.4, \quad p_- = 0.6$$

$$\mathrm{Ent}(D^{b_3}) \approx 0.97095$$

条件熵：

$$\mathrm{Ent}_B(D) \approx 0.90870$$

信息增益：

$$\text{Gain}(D, B) \approx 0.0623$$

固有值：

$$\text{IV}(B) = 1.5$$

增益率：

$$\text{GainRatio}(D, B) \approx 0.0415$$

综上

$$\text{Gain}(D, A) \approx 0.125, \quad \text{GainRatio}(D, A) \approx 0.125$$

$$\text{Gain}(D, B) \approx 0.0623, \quad \text{GainRatio}(D, B) \approx 0.0415$$

3.

ID3

比较信息增益：

$$\text{Gain}(D, A) \approx 0.125 > \text{Gain}(D, B) \approx 0.0623$$

因此选择 A。

C4.5

比较增益率：

$$\text{GainRatio}(D, A) \approx 0.125 > \text{GainRatio}(D, B) \approx 0.0415$$

因此选择 A。

CART 分类树

$$\text{Gini}(D) = 1 - \sum_{k=1}^{|\mathcal{Y}|} p_k^2$$

$$\mathrm{Gini}(D) = 0.48$$

属性 A:

$$\mathrm{Gini}(D^{a_1}) = 0.32$$
$$\mathrm{Gini}(D^{a_2}) = 0.48$$
$$\mathrm{Gini}_A(D) = 0.40$$
$$\Delta\mathrm{Gini}(A) = 0.08$$

属性 B:

$$\mathrm{Gini}(D^{b_1}) = 0.48$$
$$\mathrm{Gini}(D^{b_2}) = 0.32$$
$$\mathrm{Gini}(D^{b_3}) = 0.48$$
$$\mathrm{Gini}_B(D) = 0.44$$
$$\Delta\mathrm{Gini}(B) = 0.04$$

$$\Delta\mathrm{Gini}(A) = 0.08 > \Delta\mathrm{Gini}(B) = 0.04$$

因此选择 A。

综上

三种准则下根节点最优划分的属性均为: $\boxed{A}$

四. (25 points) 机器学习中的过拟合现象

本题以决策树与线性模型为例，探究机器学习中的过拟合现象。机器学习希望训练得到的模型在新样本上保持较好的泛化性能；如果在训练集上将模型训练得 “过好”，捕获了与任务无关的偶然特性，会导致泛化能力下降，即为过拟合。

1. 请简要总结决策树与线性模型的工作原理及其常用的缓解过拟合手段（5 points）；

2. 请使用 scikit-learn 实现决策树模型, 并扰动决策树的最大深度 max_depth, 一般来说, max_depth 的值越大, 决策树越复杂, 越容易过拟合, 实验并比较测试集精度, 讨论并分析观察到的过拟合现象等（5 points）；

3. 对决策树算法的未剪枝、预剪枝和后剪枝进行实验比较, 并进行适当的统计显著性检验（7 points）；

4. 使用 scikit-learn 实现一个线性模型：通过调整正则化强度与训练轮数，控制模型有效复杂度，实验并比较测试集精度, 讨论并分析观察到的过拟合现象等（8 points）。

注：从 UCI 机器学习库中选择 1 至 3 个数据集进行实验。

解:

1.

决策树工作原理：通过递归地选择最优特征进行数据划分，构建树形结构。常用的划分标准包括信息增益、基尼系数等。

缓解过拟合手段：预剪枝; 后剪枝; 设置叶子节点的最小样本数; 限制特征考虑的数量

线性模型工作原理：通过线性组合输入特征来预测输出，基本形式为 $y = w x + b$。

缓解过拟合手段：L1/L2 正则化（Lasso/Ridge 回归）; 早停法; 特征选择; 增加训练数据量; 降低模型复杂度

2.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False
```

```
10
11 wine = load_wine()
12 X, y = wine.data, wine.target
13 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
14
15 max_depths = range(1, 16)
16 train_scores = []
17 test_scores = []
18
19 for depth in max_depths:
20     dt = DecisionTreeClassifier(max_depth=depth, random_state=42)
21     dt.fit(X_train, y_train)
22
23     train_score = accuracy_score(y_train, dt.predict(X_train))
24     test_score = accuracy_score(y_test, dt.predict(X_test))
25
26     train_scores.append(train_score)
27     test_scores.append(test_score)
28
29     gap = train_score - test_score
30
31 plt.figure(figsize=(12, 6))
32 plt.plot(max_depths, train_scores, 'o-', label='训练集精度', linewidth=2, markersize=6)
33 plt.plot(max_depths, test_scores, 's-', label='测试集精度', linewidth=2, markersize=6)
34 plt.xlabel('决策树最大深度 (max_depth)')
35 plt.ylabel('分类精度')
36 plt.title('max_depth对过拟合的影响')
37 plt.legend()
38 plt.grid(True, alpha=0.3)
39 plt.xticks(max_depths)
40
41 best_idx = np.argmax(test_scores)
42 plt.scatter(max_depths[best_idx], test_scores[best_idx], color='red', s=100, zorder=5,
43             label=f'最佳测试精度 (max_depth={max_depths[best_idx]})')
44 plt.legend()
45
46 plt.tight_layout()
47 plt.show()
48
49 best_idx = np.argmax(test_scores)
50 print(f"\n最佳max_depth: {max_depths[best_idx]}")
51 print(f"对应的测试集精度: {test_scores[best_idx]:.4f}")
52 print(f"对应的训练集精度: {train_scores[best_idx]:.4f}")
53 print(f"精度差距: {train_scores[best_idx] - test_scores[best_idx]:.4f}")
54
55 plt.figure(figsize=(10, 5))
56 gaps = [train_scores[i] - test_scores[i] for i in range(len(train_scores))]
```

```
plt.bar(max_depths, gaps, color='orange', alpha=0.7)
plt.xlabel('决策树最大深度 (max_depth)')
plt.ylabel('训练集与测试集精度差距')
plt.title('过拟合程度随max_depth的变化')
plt.xticks(max_depths)
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()
```

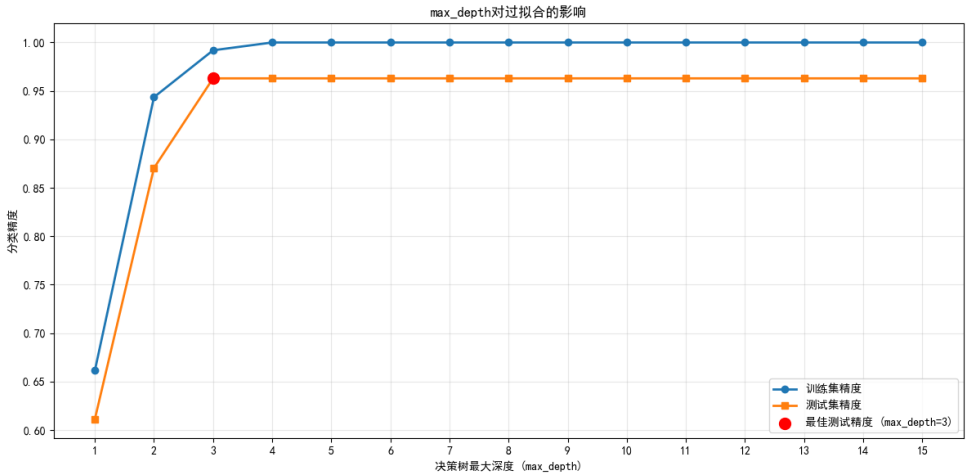

图 5: 运行结果

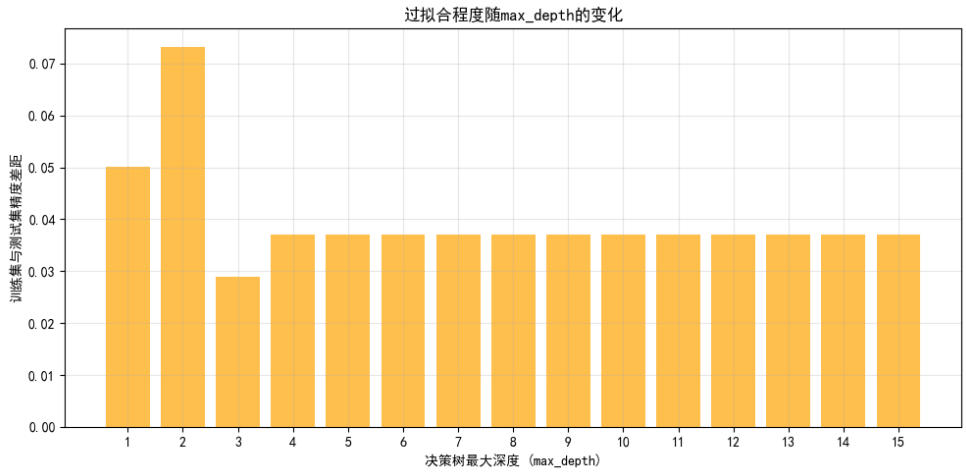

图 6: 运行结果

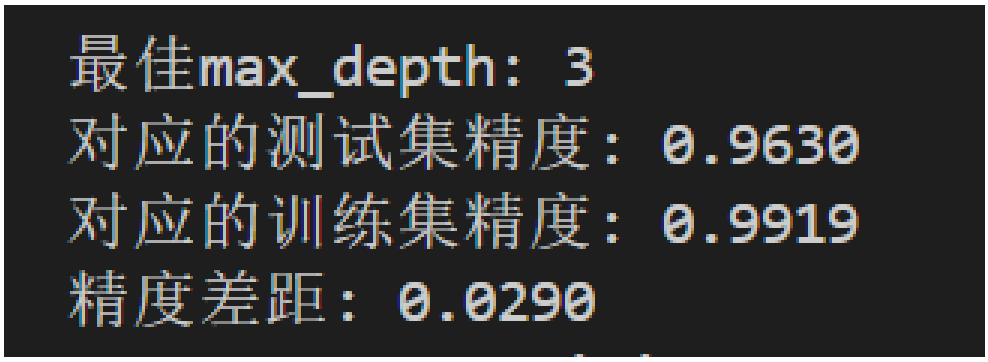

图 7: 运行结果

3.

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from scipy import stats
import pandas as pd

plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False

wine = load_wine()
X, y = wine.data, wine.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

unpruned_scores = []
for depth in range(1, 21):
    dt = DecisionTreeClassifier(max_depth=depth, random_state=42)
    scores = cross_val_score(dt, X_train, y_train, cv=5)
    unpruned_scores.append(scores.mean())

best_unpruned_depth = np.argmax(unpruned_scores) + 1
unpruned_dt = DecisionTreeClassifier(max_depth=best_unpruned_depth, random_state=42)
unpruned_dt.fit(X_train, y_train)
unpruned_train_acc = accuracy_score(y_train, unpruned_dt.predict(X_train))
unpruned_test_acc = accuracy_score(y_test, unpruned_dt.predict(X_test))

pre_pruning_params = {
    'max_depth': [3, 5, 7, 10, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 5]
```

```
}

pre_pruned_scores = []
pre_pruned_models = []

for depth in pre_pruning_params['max_depth']:
    for split in pre_pruning_params['min_samples_split']:
        for leaf in pre_pruning_params['min_samples_leaf']:
            dt = DecisionTreeClassifier(
                max_depth=depth,
                min_samples_split=split,
                min_samples_leaf=leaf,
                random_state=42
            )
            scores = cross_val_score(dt, X_train, y_train, cv=5)
            pre_pruned_scores.append(scores.mean())
            pre_pruned_models.append((depth, split, leaf))

best_pre_pruned_idx = np.argmax(pre_pruned_scores)
best_pre_params = pre_pruned_models[best_pre_pruned_idx]

pre_pruned_dt = DecisionTreeClassifier(
    max_depth=best_pre_params[0],
    min_samples_split=best_pre_params[1],
    min_samples_leaf=best_pre_params[2],
    random_state=42
)
pre_pruned_dt.fit(X_train, y_train)
pre_pruned_train_acc = accuracy_score(y_train, pre_pruned_dt.predict(X_train))
pre_pruned_test_acc = accuracy_score(y_test, pre_pruned_dt.predict(X_test))

post_pruned_dt = DecisionTreeClassifier(random_state=42)
path = post_pruned_dt.cost_complexity_pruning_path(X_train, y_train)
ccp_alphas = path.ccp_alphas

post_pruned_scores = []
post_pruned_models = []

for ccp_alpha in ccp_alphas:
    if ccp_alpha >= 0:
        dt = DecisionTreeClassifier(ccp_alpha=ccp_alpha, random_state=42)
        scores = cross_val_score(dt, X_train, y_train, cv=5)
        post_pruned_scores.append(scores.mean())
        post_pruned_models.append(ccp_alpha)

best_post_pruned_idx = np.argmax(post_pruned_scores)
best_ccp_alpha = post_pruned_models[best_post_pruned_idx]
```

```
80
81 post_pruned_dt = DecisionTreeClassifier(ccp_alpha=best_ccp_alpha, random_state=42)
82 post_pruned_dt.fit(X_train, y_train)
83 post_pruned_train_acc = accuracy_score(y_train, post_pruned_dt.predict(X_train))
84 post_pruned_test_acc = accuracy_score(y_test, post_pruned_dt.predict(X_test))
85
86 n_repeats = 50
87 unpruned_repeat_scores = []
88 pre_pruned_repeat_scores = []
89 post_pruned_repeat_scores = []
90
91 for i in range(n_repeats):
92     X_train_rep, X_test_rep, y_train_rep, y_test_rep = train_test_split(X, y, test_size
      =0.3, random_state=i)
93
94     dt_unpruned = DecisionTreeClassifier(max_depth=best_unpruned_depth, random_state=i)
95     dt_unpruned.fit(X_train_rep, y_train_rep)
96     unpruned_repeat_scores.append(accuracy_score(y_test_rep, dt_unpruned.predict(
      X_test_rep)))
97
98     dt_pre = DecisionTreeClassifier(
99         max_depth=best_pre_params[0],
100        min_samples_split=best_pre_params[1],
101        min_samples_leaf=best_pre_params[2],
102        random_state=i
103    )
104    dt_pre.fit(X_train_rep, y_train_rep)
105    pre_pruned_repeat_scores.append(accuracy_score(y_test_rep, dt_pre.predict(X_test_rep))
      )
106
107    dt_post = DecisionTreeClassifier(ccp_alpha=best_ccp_alpha, random_state=i)
108    dt_post.fit(X_train_rep, y_train_rep)
109    post_pruned_repeat_scores.append(accuracy_score(y_test_rep, dt_post.predict(X_test_rep
      )))
110
111 plt.figure(figsize=(15, 5))
112
113 plt.subplot(1, 3, 1)
114 methods = ['未剪枝', '预剪枝', '后剪枝']
115 train_accs = [unpruned_train_acc, pre_pruned_train_acc, post_pruned_train_acc]
116 test_accs = [unpruned_test_acc, pre_pruned_test_acc, post_pruned_test_acc]
117
118 x = np.arange(len(methods))
119 width = 0.35
120
121 plt.bar(x - width/2, train_accs, width, label='训练集精度', alpha=0.8)
122 plt.bar(x + width/2, test_accs, width, label='测试集精度', alpha=0.8)
```

```
123
124 plt.xlabel('剪枝方法')
125 plt.ylabel('精度')
126 plt.title('不同剪枝方法的精度比较')
127 plt.xticks(x, methods)
128 plt.legend()
129 plt.grid(True, alpha=0.3)
130
131 plt.subplot(1, 3, 2)
132 overfitting_gaps = [train_acc - test_acc for train_acc, test_acc in zip(train_accs,
        test_accs)]
133 plt.bar(methods, overfitting_gaps, color='orange', alpha=0.7)
134 plt.xlabel('剪枝方法')
135 plt.ylabel('训练集-测试集精度差距')
136 plt.title('过拟合程度比较')
137 plt.grid(True, alpha=0.3)
138
139 plt.subplot(1, 3, 3)
140 data = [unpruned_repeat_scores, pre_pruned_repeat_scores, post_pruned_repeat_scores]
141 plt.boxplot(data, labels=methods)
142 plt.xlabel('剪枝方法')
143 plt.ylabel('测试集精度')
144 plt.title('模型稳定性比较 (50次重复实验)')
145 plt.grid(True, alpha=0.3)
146
147 plt.tight_layout()
148 plt.show()
149
150 print("\n" + "="*60)
151 print("模型稳定性分析")
152 print("="*60)
153
154 stability_data = {
155     '方法': methods,
156     '平均精度': [np.mean(scores) for scores in data],
157     '精度标准差': [np.std(scores) for scores in data],
158     '最小精度': [np.min(scores) for scores in data],
159     '最大精度': [np.max(scores) for scores in data]
160 }
161
162 stability_df = pd.DataFrame(stability_data)
163 print(stability_df.round(4))
164
165 print("\n结论:")
166 best_method_idx = np.argmax([np.mean(scores) for scores in data])
167 print(f"1. 最佳性能方法: {methods[best_method_idx]} (平均精度: {np.mean(data[
        best_method_idx]):.4f})")
```

```
168
169 most_stable_idx = np.argmin([np.std(scores) for scores in data])
170 print(f"2. 最稳定方法: {methods[most_stable_idx]} (标准差: {np.std(data[most_stable_idx])
        :.4f})")
171
172 least_overfitting_idx = np.argmin(overfitting_gaps)
173 print(f"3. 过拟合控制最佳方法: {methods[least_overfitting_idx]} (过拟合程度: {
        overfitting_gaps[least_overfitting_idx]:.4f})")
```

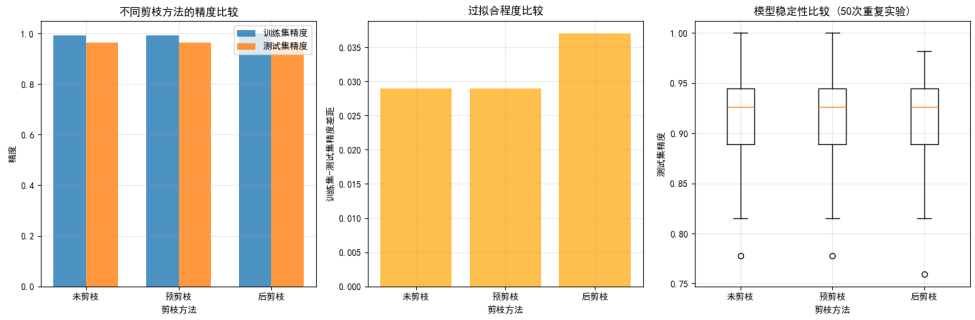

图 8: 运行结果

```
============================================================
模型稳定性分析
============================================================
    方法    平均精度   精度标准差    最小精度    最大精度
0  未剪枝  0.9156  0.0460  0.7778  1.0000
1  预剪枝  0.9156  0.0460  0.7778  1.0000
2  后剪枝  0.9130  0.0466  0.7593  0.9815

结论:
1. 最佳性能方法: 未剪枝 (平均精度: 0.9156)
2. 最稳定方法: 未剪枝 (标准差: 0.0460)
3. 过拟合控制最佳方法: 未剪枝 (过拟合程度: 0.0290)
```

图 9: 运行结果

4.

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
from sklearn.linear_model import SGDClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score

plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False

wine = load_wine()
X, y = wine.data, wine.target

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.3,
    random_state=42)

alpha_values = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100]
max_iters = 1000

alpha_train_scores = []
alpha_test_scores = []

for alpha in alpha_values:
    model = SGDClassifier(
        loss='log_loss',
        penalty='l2',
        alpha=alpha,
        max_iter=max_iters,
        random_state=42
    )

    model.fit(X_train, y_train)

    train_score = accuracy_score(y_train, model.predict(X_train))
    test_score = accuracy_score(y_test, model.predict(X_test))

    alpha_train_scores.append(train_score)
    alpha_test_scores.append(test_score)

    overfitting = train_score - test_score

```

```
iter_values = [10, 50, 100, 200, 500, 1000, 2000, 5000]
alpha_fixed = 0.01

iter_train_scores = []
iter_test_scores = []

for n_iter in iter_values:
    model = SGDClassifier(
        loss='log_loss',
        penalty='l2',
        alpha=alpha_fixed,
        max_iter=n_iter,
        random_state=42
    )

    model.fit(X_train, y_train)

    train_score = accuracy_score(y_train, model.predict(X_train))
    test_score = accuracy_score(y_test, model.predict(X_test))

    iter_train_scores.append(train_score)
    iter_test_scores.append(test_score)

    overfitting = train_score - test_score

plt.figure(figsize=(12, 5))

# 1. 正则化强度的影响
plt.subplot(1, 2, 1)
plt.semilogx(alpha_values, alpha_train_scores, 'o-', label='训练集精度', linewidth=2)
plt.semilogx(alpha_values, alpha_test_scores, 's-', label='测试集精度', linewidth=2)
plt.xlabel('正则化强度 (alpha)')
plt.ylabel('分类精度')
plt.title('正则化强度对精度的影响')
plt.legend()
plt.grid(True, alpha=0.3)

# 2. 训练轮数的影响
plt.subplot(1, 2, 2)
plt.semilogx(iter_values, iter_train_scores, 'o-', label='训练集精度', linewidth=2)
plt.semilogx(iter_values, iter_test_scores, 's-', label='测试集精度', linewidth=2)
plt.xlabel('训练轮数')
plt.ylabel('分类精度')
plt.title('训练轮数对精度的影响')
plt.legend()
plt.grid(True, alpha=0.3)

```

```
92 plt.tight_layout()
93 plt.show()
94
95 print("\n过拟合现象分析:")
96 print("=" * 50)
97
98 best_alpha_idx = np.argmax(alpha_test_scores)
99 best_iter_idx = np.argmax(iter_test_scores)
100
101 print(f"1. 最佳正则化强度: alpha={alpha_values[best_alpha_idx]}")
102 print(f"   测试集精度: {alpha_test_scores[best_alpha_idx]:.4f}")
103 print(f"   过拟合程度: {alpha_train_scores[best_alpha_idx] - alpha_test_scores[
    best_alpha_idx]:.4f}")
104
105 print(f"\n2. 最佳训练轮数: max_iter={iter_values[best_iter_idx]}")
106 print(f"   测试集精度: {iter_test_scores[best_iter_idx]:.4f}")
107 print(f"   过拟合程度: {iter_train_scores[best_iter_idx] - iter_test_scores[best_iter_idx
    ]:.4f}")
108
109 print(f"\n3. 过拟合模式:")
110 print("   - 正则化太弱 (alpha < 0.01): 训练精度高, 测试精度低, 明显过拟合")
111 print("   - 正则化太强 (alpha > 1): 训练和测试精度都低, 欠拟合")
112 print("   - 训练轮数太少 (< 100): 模型未收敛, 欠拟合")
113 print("   - 训练轮数太多 (> 2000): 可能过度拟合训练数据")
```

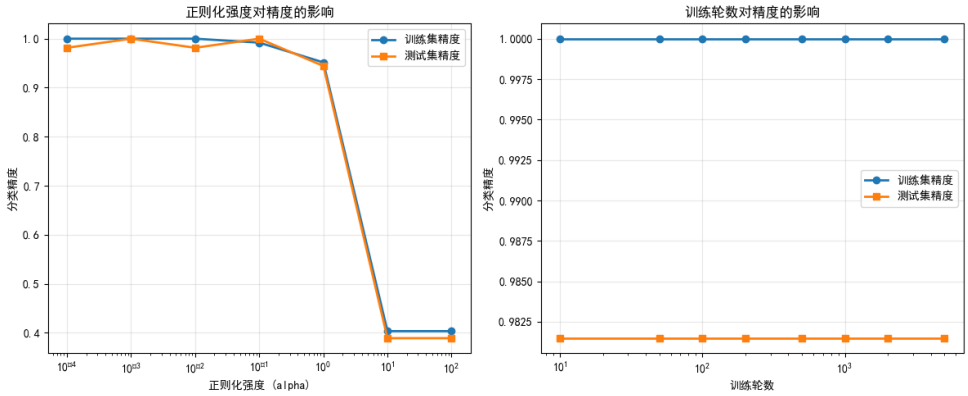

图 10: 运行结果

```
过拟合现象分析:
==================================================
1. 最佳正则化强度: alpha=0.001
   测试集精度: 1.0000
   过拟合程度: 0.0000

2. 最佳训练轮数: max_iter=10
   测试集精度: 0.9815
   过拟合程度: 0.0185

3. 过拟合模式:
   - 正则化太弱 (alpha < 0.01): 训练精度高，测试精度低，明显过拟合
   - 正则化太强 (alpha > 1): 训练和测试精度都低，欠拟合
   - 训练轮数太少 (< 100): 模型未收敛，欠拟合
   - 训练轮数太多 (> 2000): 可能过度拟合训练数据
```

图 11: 运行结果