一、实验目的

- 学会理解数据并对数据进行预处理；
- 理解决策树的原理并掌握其构建方法；

二、实验环境

- Jupyter Notebook
- PyCharm 2022.2 (Community Edition)
- Python 3.9.12

三、实验内容

- 熟悉 Pandas 库的安装和使用，并对数据进行预处理和相关分析；
- 编写代码实现一种决策树算法（不允许使用 sklearn 等现有开源库），对测试集数据进行预测（分类任务，判断年收入是否大于 50K 美元）；

四、部分文件说明

- lab2-baseline.ipynb：不对决策树进行剪枝的代码（同时包含了运行结果）
- lab2-baseline.pdf：将 lab2-baseline.ipynb 输出为 pdf 格式以便于查看
- lab2-optimization.py：对决策树进行剪枝优化的代码
- test_adult_processed.csv：进行数据预处理后的测试集数据
- train_adult_processed.csv：进行数据预处理后的训练集数据
- 测试集真实值与预测值的对比（无剪枝）.csv：预测结果文件
- 在不同最大层数限制下的预测准确度.txt：进行不同程度剪枝后的预测准确度

五、实验过程

1. **数据预处理**方案的设计

数据预处理方面，我参考了题目所给的**代码模板**中已经处理的部分进行设计。

首先，将训练集和测试集的数据从文件中加载出来，并使用 pandas 库提供的 DataFrame 进行存储和操作（模板代码已实现）。

模板代码删除了训练中用处不大的属性 fnlwgt 和 educationNum，显然对**测试集**在进行预测的时候也用不到这两列属性，所以我在**测试集**中也把这两列属性给删去了。

模板代码也删除了**训练集**的重复记录和缺失值（本实验的数据集将缺失值标记为问号“？”而不是为空，因此使用 dropna()方法无法去除，只能使用正则表达式匹配的方法找到含有“？”的数据项进行清除）。我也仿照模板代码的方式对其它属性进行了去除缺失（异常）值处理。但是，**测试集**是需要我们去预测结果的，我们不应该随意增删测试集数据的数目，所以不能对**测试集**进行同样的去除重复值、去除缺失（异常）值操作。

对于连续型变量的处理，模板代码的做法是对**训练集**的年龄特征进行分箱操作。为了保证训练集与测试集数据的一致性，我对**测试集**也进行了**相同的分箱处理**。对于其它的连续型变量，由于它们的取值都不多，我决定使用实验指导书中介绍的二分法进行处理，即在构建决策树的过程中选取每两个点的中点进行划分，找到其中最优的。

对于离散型变量的处理，模板代码的操作是对**训练集**的 workclass 属性进行编号（同时合并了部分取值），因此我也对**测试集**的 workclass 属性使用同样的方法进行编号。对于其它离散型变量，我都是仿照模板代码的做法，直接进行编号。其中对于教育属性，我进行了合并，如将小学 1-4 年级和小学 5-6 年级归为一类。

训练集和测试集数据预处理的结果可以查看文件 train_adult_processed.csv 以及文件 test_adult_processed.csv。

2. 决策树构建方案的设计

本次实验我构建的是 **CART 决策树**，即在生长过程中使用基尼系数作为判断依据。在分裂决策树的过程中，我的设计是遍历所有属性的所有可能取值，假

设使用属性、取值进行划分，求出划分后的**基尼系数**，挑选出其中最优的组合，然后划分数据集、构建子树。

对于决策树递归构建的停止条件，我设置了两个，一个是当前节点所有样本标签相同，这说明样本已经被分“纯”；另一个是当前训练集所有的特征都已经被使用完毕，也就是说**使用任何特征都已经无法再划分当前样本了**。

在利用构建好的决策树预测某一数据时，做法是按照决策树的构建规则进行搜索，每次跳转到左孩子节点或右孩子节点，最终找到当前待预测数据对应的**叶子节点**，将该叶子节点存储的标签值作为预测结果。

（1） 基尼系数的计算

```
def calc_gini(df):
    """
    计算数据集的基尼指数
    :param df: 数据集
    :return: 基尼指数
    """
    p0 = 0
    n = 0
    for num in df['income']:
        if num == 0:
            p0 += 1
        n += 1
    p0 = p0 / n
    p1 = 1 - p0
    return 1 - p0 * p0 - p1 * p1
```

（2） 按照特定的属性、属性值划分数据集

```
def split_dataset(df, index, value):
    """
    按照给定的列划分数据集
    :param df: 原始数据集
    :param index: 指定特征的列索引
    :param value: 指定特征的值
    :return: 切分后的数据集(left_df, right_df)
    """
    # 将数据集划分为两半，分发给左子树和右子树
    # index 对应离散型特征时，左子树为符合 value 的子集，右子树为不符合 value
的子集
    # index 对应连续型特征时，左子树为小于等于 value 的子集，右子树为大于 value
的子集
    feature = columns[index]
```

```
    if feature in discrete_column:
        left_df = df[df[feature] == value]
        right_df = df[df[feature] != value]
    else:
        left_df = df[df[feature] <= value]
        right_df = df[df[feature] > value]
    return left_df, right_df
```

（3） 分裂决策树

```
def choose_best_feature_to_split(df):
    """
    选择最好的特征进行分裂
    :param df: 数据集
    :return: best_value:(分裂特征的 index, 特征的值), best_df:(分裂后的左右子树数据集), min_gini:(选择该属性分裂的最小基尼指数)
    """
    best_value = ()
    min_gini = calc_gini(df)
    best_df = ()
    for index in range(len(columns) - 1): # 最后一列是 income，因此要减 1
        feature = columns[index]
        for val in set(df[feature].values):
            left_df, right_df = split_dataset(df, index, val)
            left_size = len(left_df)
            right_size = len(right_df)
            if left_size == 0 or right_size == 0:
                continue
            total_size = left_size + right_size
            left_gini = calc_gini(left_df)
            right_gini = calc_gini(right_df)
            new_gini = left_gini * left_size / total_size + right_gini * right_size / total_size
            if new_gini < min_gini:
                min_gini = new_gini
                best_value = index, val
                best_df = left_df, right_df
    return best_value, best_df, min_gini
```

（4） 构建决策树

```
def build_decision_tree(df):
    """
    构建 CART 树
    :param df: 数据集
    :return: CART 树
    """
```

```
    best_value, best_df, min_gini = choose_best_feature_to_split(df)
    # CART 树表示为[leaf_flag, label, left_tree, right_tree,
best_value]，其中 leaf_flag 标记是否为叶子
    if len(set(df['income'])) == 1: # 若 income 的取值只有一种，说明已分
“纯”
        cart = np.array([1, list(df['income'])[0], None, None, ()],
dtype=object)
        return cart # 递归结束情况 1：若当前集合的所有样本标签相等,即样本已被
分"纯",则可以返回该标签值作为一个叶子节点
    elif best_value == (): # 若 best_value 为()，说明已经没有可用的特征
        if sum(df['income']) > (len(df['income']) - sum(df['income'])):
            label = 1
        else:
            label = 0
        cart = np.array([1, label, None, None, ()], dtype=object)
        return cart # 递归结束情况 2：若当前训练集的所有特征都被使用完毕,当前
无可用特征但样本仍未分"纯"，则返回样本最多的标签作为结果
    else:
        left_tree = build_decision_tree(best_df[0])
        right_tree = build_decision_tree(best_df[1])
        cart = np.array([0, -1, left_tree, right_tree, best_value],
dtype=object)
        return cart
```

（5） 利用决策树进行预测

```
def classify(cart, df_row):
    """
    用训练好的决策树进行分类
    :param cart:决策树模型
    :param df_row: 一条测试样本
    :return: 预测结果
    """
    while cart[0] != 1:
        index, value = cart[4]
        feature = columns[index]
        if feature in discrete_column:
            if df_row[feature] == value:
                cart = cart[2]
            else:
                cart = cart[3]
        else:
            if df_row[feature] <= value:
                cart = cart[2]
            else:
                cart = cart[3]
```

```
    return cart[1]
```

3. 对决策树进行剪枝优化的设计

上一部分在构建决策树时没有进行剪枝处理，也没有限制决策树的层数，最终得到模型在测试集上的预测准确度约为 **0.838216**。为了优化模型，我进行了较为简单的**预剪枝处理**，也就是限制决策树构建的高度以防止过拟合，减少噪声数据的影响。具体来说，我为决策树设置不同的最大层数，并同时评估其在训练集、预测集上的表现。最终，我发现进行预剪枝确实能提高预测准确度，最大层数为 **12 层左右**的时候效果最佳，准确度能提升至约 **0.856**。

为进行预剪枝处理，构建决策树的代码需要修改成如下所示。

```
def build_decision_tree(df, layer, max_layer):
    """
    构建 CART 树
    :param df: 数据集
    :param layer: 当前所在层数(从 0 开始计算)
    :param max_layer: 剪枝时所允许的最大层数(为-1 时则不减枝)
    :return: CART 树
    """
    best_value, best_df, min_gini = choose_best_feature_to_split(df)
    # CART 树表示为[leaf_flag, label, left_tree, right_tree, best_value,
layer]
    # 其中 leaf_flag 标记是否为叶子
    if len(set(df['income'])) == 1:  # 若 income 的取值只有一种，说明已分
“纯”
        cart = np.array([1, list(df['income'])[0], None, None, (),
layer], dtype=object)
        return cart  # 递归结束情况 1: 若当前集合的所有样本标签相等,即样本已
被分"纯",则可以返回该标签值作为一个叶子节点
    elif best_value == () or (max_layer != -1 and layer >= max_layer):
        # 若 best_value 为(), 说明已经没有可用的特征; 若 layer >= max_layer
且 max_layer 不为-1, 说明需要剪枝
        if sum(df['income']) > (len(df['income']) - sum(df['income'])):
            label = 1
        else:
            label = 0
        cart = np.array([1, label, None, None, (), layer], dtype=object)
        return cart  # 递归结束情况 2: 若当前训练集的所有特征都被使用完毕或需
要剪枝, 则返回样本最多的标签作为结果
    else:
        left_tree = build_decision_tree(best_df[0], layer + 1,
max_layer)
```

```
        right_tree = build_decision_tree(best_df[1], layer + 1,
max_layer)
        cart = np.array([0, -1, left_tree, right_tree, best_value,
layer], dtype=object)
        return cart
```

六、实验结果

（一） 未对决策树进行剪枝优化的结果

详细运行结果可查看文件 lab2-baseline.ipynb 以及文件 lab2-baseline.pdf。


5. 运行模型

用测试集评估模型的准确性

```
In [39]: cart = load_decision_tree() # 加载模型

In [40]: test_list = df_test_set['income'].to_numpy()
         pred_list = predict(cart, df_test_set)

In [41]: acc = calc_acc(pred_list, test_list)

In [42]: acc
Out[42]: 0.8382163257785148
```

另外，对于测试集中每一个样本的**具体预测结果**，可以查看文件“测试集真实值与预测值的对比（无剪枝）.csv”。

（二） 对决策树进行剪枝优化的结果

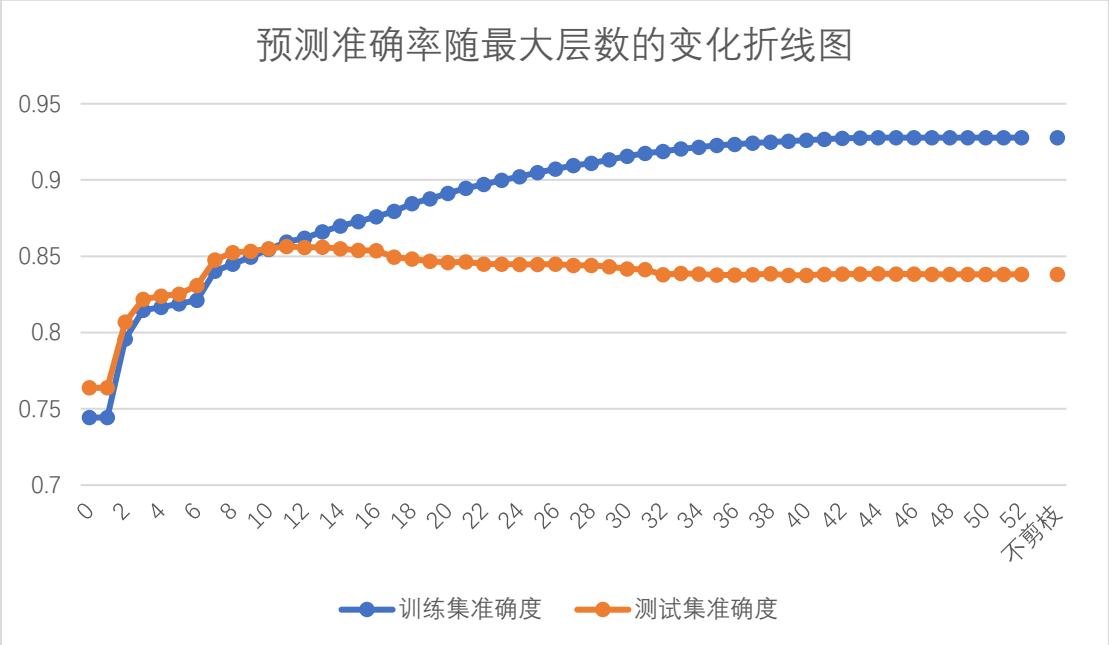
下图结果是对程序 lab2-optimization.py 进行执行后得到的，完整数据结果可查看文件“在不同最大层数限制下的预测准确度.txt”。

```
不对决策树的高度进行限制：
此时在训练集上的准确率为0.9277802557240559，在测试集上的准确率为0.8382163257785148

限制决策树的最大层数为0(从0开始编号)：
此时在训练集上的准确率为0.744275944097532，在测试集上的准确率为0.7637737239727289

限制决策树的最大层数为1(从0开始编号)：
此时在训练集上的准确率为0.744275944097532，在测试集上的准确率为0.7637737239727289

限制决策树的最大层数为2(从0开始编号)：
此时在训练集上的准确率为0.7957181088314005，在测试集上的准确率为0.8068300472943922

限制决策树的最大层数为3(从0开始编号)：
此时在训练集上的准确率为0.8146743978590544，在测试集上的准确率为0.8218782630059579

限制决策树的最大层数为4(从0开始编号)：
此时在训练集上的准确率为0.8166443651501636，在测试集上的准确率为0.8239665868189915

限制决策树的最大层数为5(从0开始编号)：
此时在训练集上的准确率为0.8189116859946476，在测试集上的准确率为0.8251335913027456

限制决策树的最大层数为6(从0开始编号)：
此时在训练集上的准确率为0.8212161760333035，在测试集上的准确率为0.83078434985566

限制决策树的最大层数为7(从0开始编号)：
此时在训练集上的准确率为0.8402468034493012，在测试集上的准确率为0.847613782937166
```

最终得到测试集的最高准确率约为 **0.8565199**，此时最大层数为 **11**。

```
限制决策树的最大层数为11(从0开始编号)：
此时在训练集上的准确率为0.8593146000594707，在测试集上的准确率为0.8565198697868681
```

绘制训练集、测试集的**预测准确率**随最大层数的**变化折线图**，如下所示。



可以发现，随着层数增加，训练集的预测准确度逐渐提升，测试集的预测准确度先上升后下降，这可能是**过拟合**导致的。