1 决策树算法

[1]:
```
import numpy as np
import pandas as pd
import json
```

1.1 正确地读取数据

注意原始数据文件的格式，对其进行正确地处理后读入两个 DataFrame：`adult_data_df` 是训练集，`adult_test_df` 是测试集。DataFrame 中名为 “50K” 的列为标签（即分类）。

[2]:
```
col_names = ['age', 'workclass', 'fnlwgt', 'education', 'education-num',␣
↪'marital-status', 'occupation', 'relationship', 'race', 'sex',␣
↪'capital-gain', 'capital-loss', 'hours-per-week', 'native-country', '50K']
adult_data_df = pd.read_csv('dataset/adult.data', index_col=False, header=None,␣
↪names=col_names, sep=', ', engine='python')#.drop(['fnlwgt'], axis=1)
adult_data_df
```

[2]:
```
       age         workclass  fnlwgt   education  education-num  \
0       39         State-gov   77516   Bachelors             13
1       50  Self-emp-not-inc   83311   Bachelors             13
2       38           Private  215646     HS-grad              9
3       53           Private  234721        11th              7
4       28           Private  338409   Bachelors             13
...    ...               ...     ...         ...            ...
32556   27           Private  257302  Assoc-acdm             12
32557   40           Private  154374     HS-grad              9
32558   58           Private  151910     HS-grad              9
32559   22           Private  201490     HS-grad              9
32560   52      Self-emp-inc  287927     HS-grad              9

           marital-status         occupation   relationship   race     sex  \
0           Never-married       Adm-clerical  Not-in-family  White    Male
1      Married-civ-spouse    Exec-managerial        Husband  White    Male
2                Divorced  Handlers-cleaners  Not-in-family  White    Male
3      Married-civ-spouse  Handlers-cleaners        Husband  Black    Male
4      Married-civ-spouse     Prof-specialty           Wife  Black  Female
...                   ...                ...            ...    ...     ...
32556  Married-civ-spouse       Tech-support           Wife  White  Female
32557  Married-civ-spouse  Machine-op-inspct        Husband  White    Male
32558             Widowed       Adm-clerical      Unmarried  White  Female
32559       Never-married       Adm-clerical      Own-child  White    Male
32560  Married-civ-spouse    Exec-managerial           Wife  White  Female

       capital-gain  capital-loss  hours-per-week native-country    50K
0              2174             0              40  United-States  <=50K
1                 0             0              13  United-States  <=50K
2                 0             0              40  United-States  <=50K
```

```
3                0            0            40  United-States   <=50K
4                0            0            40           Cuba   <=50K
...            ...          ...           ...            ...     ...
32556            0            0            38  United-States   <=50K
32557            0            0            40  United-States    >50K
32558            0            0            40  United-States   <=50K
32559            0            0            20  United-States   <=50K
32560        15024            0            40  United-States    >50K

[32561 rows x 15 columns]
```

[3]:
```python
adult_test_df = pd.read_csv('dataset/adult.test', skiprows=[0], index_col=False, header=None, names=col_names, sep=', ', engine='python')#.drop(['fnlwgt'], axis=1)
adult_test_df['50K'] = adult_test_df['50K'].map(lambda x: x[:-1])  # 去除行末的点
adult_test_df
```

[3]:
```
       age     workclass  fnlwgt     education  education-num  \
0       25       Private  226802          11th              7
1       38       Private   89814       HS-grad              9
2       28     Local-gov  336951    Assoc-acdm             12
3       44       Private  160323  Some-college             10
4       18             ?  103497  Some-college             10
...    ...           ...     ...           ...            ...
16276   39       Private  215419     Bachelors             13
16277   64             ?  321403       HS-grad              9
16278   38       Private  374983     Bachelors             13
16279   44       Private   83891     Bachelors             13
16280   35  Self-emp-inc  182148     Bachelors             13

           marital-status         occupation   relationship  \
0           Never-married  Machine-op-inspct      Own-child
1      Married-civ-spouse    Farming-fishing        Husband
2      Married-civ-spouse    Protective-serv        Husband
3      Married-civ-spouse  Machine-op-inspct        Husband
4           Never-married                  ?      Own-child
...                   ...                ...            ...
16276            Divorced     Prof-specialty  Not-in-family
16277             Widowed                  ? Other-relative
16278  Married-civ-spouse     Prof-specialty        Husband
16279            Divorced       Adm-clerical      Own-child
16280  Married-civ-spouse    Exec-managerial        Husband

                      race    sex  capital-gain  capital-loss  hours-per-week  \
0                    Black   Male             0             0              40
1                    White   Male             0             0              50
2                    White   Male             0             0              40
```

```
3                Black    Male          7688          0          40
4                White  Female             0          0          30
...                ...     ...           ...        ...         ...
16276            White  Female             0          0          36
16277            Black    Male             0          0          40
16278            White    Male             0          0          50
16279   Asian-Pac-Islander  Male        5455          0          40
16280            White    Male             0          0          60

      native-country    50K
0      United-States  <=50K
1      United-States  <=50K
2      United-States   >50K
3      United-States   >50K
4      United-States  <=50K
...              ...    ...
16276  United-States  <=50K
16277  United-States  <=50K
16278  United-States  <=50K
16279  United-States  <=50K
16280  United-States   >50K

[16281 rows x 15 columns]
```

1.2 补充缺失值

通过对数据的基本观察得知，缺失值所在的列均为离散属性，因此只需要对离散缺失值进行补全即可，本例数据集上无需考虑连续型数据的补全。我采用的方法是使用该列出现次数最多的值（即众数）代替缺失值。

[4]:
```
# 补充缺失值，
print('[adult.data]')
mode_df = adult_data_df.mode()  # 众数
for col in adult_data_df:
    if '?' in adult_data_df[col].tolist():
        missing_count = adult_data_df[col].value_counts()['?']   # 缺失值的个数
        adult_data_df[col] = adult_data_df[col].replace('?', mode_df[col][0])
        print('{}: {} missing values are replaced with "{}"'.format(col,
 ↪missing_count, mode_df[col][0]))

print('-------------------------------')
print('[adult.test]')
mode_df = adult_test_df.mode()  # 众数
for col in adult_test_df:
    if '?' in adult_test_df[col].tolist():
        missing_count = adult_test_df[col].value_counts()['?']   # 缺失值的个数
        adult_test_df[col] = adult_test_df[col].replace('?', mode_df[col][0])
```

```
        print('{}: {} missing values are replaced with "{}"'.format(col,␣
↪missing_count, mode_df[col][0]))
```

```
[adult.data]
workclass: 1836 missing values are replaced with "Private"
occupation: 1843 missing values are replaced with "Prof-specialty"
native-country: 583 missing values are replaced with "United-States"
-------------------------------
[adult.test]
workclass: 963 missing values are replaced with "Private"
occupation: 966 missing values are replaced with "Prof-specialty"
native-country: 274 missing values are replaced with "United-States"
```

1.3 处理连续型变量

需要将连续型变量离散化，离散化方法是二分法（bi-partition），选取使得划分后信息增益最大的点作为划分点。方法详见 "西瓜书" 第 4.4 节。

[5]:
```
def entropy(df):
    """ 计算信息熵。
    Args:
        df: 要计算信息熵的二分类数据集。
    Returns:
        信息熵值。
    """
    try:
        q = df['50K'].value_counts()['<=50K'] / len(df['50K'])  # 正样本的概率
    except:
        q = 0
    if q == 0 or q == 1:
        return 0  # 约定
    else:
        return -(q * np.log2(q) + (1-q) * np.log2(1-q))

def informationGain(df, attribute):
    """ 计算信息增益。
    Args:
        df: 数据集。
        attribute: 选取的属性。
    Returns:
        信息增益值。
    """
    remainder = 0  # 累积条件熵
    # 对指定属性的每个取值 value
    for value in df[attribute].unique():
        sub_df = df[df[attribute]==value]
        remainder += len(sub_df)/len(df) * entropy(sub_df)
    return entropy(df) - remainder  # 信息熵 - 条件熵
```

[6]:

```python
continuous_attrs = ['age', 'fnlwgt', 'education-num', 'capital-gain', 'capital-loss', 'hours-per-week']  # 连续型属性

for attr in continuous_attrs:
    partition_point = -1
    max_ig = 0

    # 在训练集上尝试以每个值进行划分，选出信息增益最大的那个划分点
    for value in sorted(list(adult_data_df[attr].unique())):
        adult_data_df['temp'] = adult_data_df[attr].map(lambda x: '+' if x>value else '-')  # 大于划分点表示为'+'，小于等于划分点表示为'-'
        current_ig = informationGain(adult_data_df, 'temp')  # 计算当前划分的信息增益
        if current_ig >= max_ig:
            partition_point = value
            max_ig = current_ig
    adult_data_df.drop(['temp'], axis=1, inplace=True)  # 删掉临时属性列

    # 用同样的划分点离散化训练集和测试集
    adult_data_df[attr] = adult_data_df[attr].map(lambda x: '{}+'.format(partition_point) if x>partition_point else '{}-'.format(partition_point))
    adult_test_df[attr] = adult_test_df[attr].map(lambda x: '{}+'.format(partition_point) if x>partition_point else '{}-'.format(partition_point))
    print(attr, partition_point)  # debug

# 保存离散化后的数据集，方便下次使用
adult_data_df.to_csv('dataset/discretized_adult.data', index=False)
adult_test_df.to_csv('dataset/discretized_adult.test', index=False)
```

```
age 27
fnlwgt 209912
education-num 12
capital-gain 6849
capital-loss 1816
hours-per-week 41
```

上面步骤中，对 `fnlwgt` 属性的处理很慢。然而实验结果表明，即使不考虑该属性，对模型准确性也不会产生明显影响。

[7]:

```python
# 从文件中读取预处理过的数据集
adult_data_df = pd.read_csv('dataset/discretized_adult.data')
adult_test_df = pd.read_csv('dataset/discretized_adult.test')
```

1.4 编码

为了方便表示，可以考虑将离散属性编码为整数。但在本例中是一个可选的步骤，直接用字符串表示的属性值表示属性取值同样可以，且具有更高的可读性（但可能略微损失少许性能，因为处理字符串比处理整数稍慢）。

我省略了编码这一步骤，直接用属性字符串值表示节点内容。

1.5 构建决策树

构建决策树的过程参考了“西瓜书”第 4 章图 4.2 的伪代码，并做了一些修改。修改了当样例为空时的行为，并增加了一个简单的剪枝条件。表示样例、属性的数据结构均使用 DataFrame。决策树表示为字典，字典的键由树节点、树边交替构成。

方便起见，我将训练好的决策树保存为 `tree_id3.json` 文件。

[8]:
```
# 训练决策树
def treeGenerate(df, mostImportant):
    """ 生成一棵完整的决策树。

    Args:
        df: 训练集，其中标签是名为'50K' 的列。
        mostImportant: 获得最优划分属性的函数。
    Returns:
        由字典表示的树。字典的键由树节点、树边交替构成；字典的值是子树或叶节点。
    """
    # 若所有样本属于同一类别，则返回该类别
    if len(df['50K'].unique()) == 1:
        return df['50K'].iloc[0]  # 叶节点，返回标签
    # 若属性集为空，返回样本数最多的类
    if len(df.columns) == 1:
        return df['50K'].value_counts().index[0]  # 叶节点，返回居右最多样本数的
标签
    if len(df) < 200:  # 剪枝
        return df['50K'].value_counts().index[0]

    best_attribute = mostImportant(df)  # 最优划分属性
    tree = {best_attribute: {}}         # 准备构造当前节点

    # 对原始数据集中该属性的所有取值（注意这里用的不是子集，否则会导致树不完整）
    for value in adult_data_df[best_attribute].unique():
        next_df = df[df[best_attribute]==value].drop([best_attribute], axis=1)
        if len(next_df) == 0:  # 该取值的样本集为空
            tree[best_attribute][value] =  df['50K'].value_counts().index[0]  #␣
↪此处本应直接返回叶节点，但实验结果表明继续分枝效果更好，且对性能影响很小
        else:  # 递归
            tree[best_attribute][value] = treeGenerate(next_df, mostImportant)
    return tree  # 返回子树
```

[9]:
```
def id3(df):
    """ID3 算法划分属性。
```

```
    Args:
        df: 要进行属性划分的数据集，其中标签是名为'50K' 的列。调用条件保证 df 至少有两列（包括标签列）。
    Returns:
        一个属性，按该属性划分可以使信息增益最大。
    """
    attributes = list(df.columns)
    attributes.remove('50K')  # 标签列不是属性，去除后 attributes 中至少有一个属性

    max_ig = 0
    best_attribute = attributes[0]
    for attribute in attributes[1:]:
        current_ig = informationGain(df, attribute)
        if current_ig > max_ig:
            best_attribute = attribute
            max_ig = current_ig
    return best_attribute
```

[10]:
```
tree_id3 = treeGenerate(adult_data_df, id3)
```

[11]:
```
# 把决策树保存为 JSON 文件
with open('tree_id3.json', 'w') as f:
    json.dump(tree_id3, f)
```

1.6 验证

在测试集上检验决策树模型的准确率。

[12]:
```
def testSample(sample, tree):
    """ 测试一个样本的正确性。
    Args:
        sample: 一个待测试样本。
        tree: 决策树模型。
    Returns:
        True 表示正确，False 表示错误。
    """
    while type(tree) == type({}):  # 子树类型一旦不是字典则表示到达叶节点
        attribute = list(tree.keys())[0]
        tree = tree[attribute][sample[attribute]]
    return tree == sample['50K']

def test(df, tree):
    """ 测试给定数据集上的预测正确率。
    Args:
        df: 测试数据集。
        tree: 决策树模型。
    Returns:
        预测正确率。
```

```
    """
    correct_count = 0
    for i in range(len(df)):
        if testSample(df.iloc[i], tree):
            correct_count += 1
    return correct_count / len(df)
```

[13]:
```
# 从 JSON 文件中读取决策树
with open('tree_id3.json') as f:
    tree_id3 = json.load(f)
```

[14]:
```
# 训练集准确率（供参考）
test(adult_data_df, tree_id3)
```

[14]: 0.8528914959614262

[15]:
```
# 测试集准确率
test(adult_test_df, tree_id3)
```

[15]: 0.8469381487623611

最终，该决策树模型在测试集上的准确率为 84.7%。