第十一周学习笔记—决策树汇总

2022-06-23

第一章　ID3 决策树

根据划分方法不同可以分为 ID3、C4.5、CART 三种决策树。

1.1　信息熵

决策树算法的关键在于如何选择最优划分属性。

一般而言，我们希望决策树的分支节点所包含的样本尽可能属于同一类别，即其纯度越高越好。

通常，使用信息熵（information entropy）来作为度量样本纯度的标准，计算公式为：

$$Ent(D) = -\sum_{n=1}^{|y|} p_k \log_2 p_k$$

其中 $|y|$ 表示有几类，p_k 表示第 k 类样本的占比。

信息熵值越小，纯度则越高

举个例子：对于二分类，假设现在划分节点使得样本分类各占一半，则根据上述公式，信息熵为:

$$Ent = -0.5 \times \log_2(0.5) \times 2 = 1$$

而当划分节点使得样本按照 91 开分为 2 类时，根据上述公式，信息熵为:

$$Ent = -0.9 \times \log_2(0.9) - 0.1 \times \log_2(0.1) = 0.469$$

根据我们的定义，Ent 的值越小纯度越高，即当划分数据越倾向一类越好，当数据均分时纯度较低。

我们可以证明如下：

随机变量 X 的熵定义为:

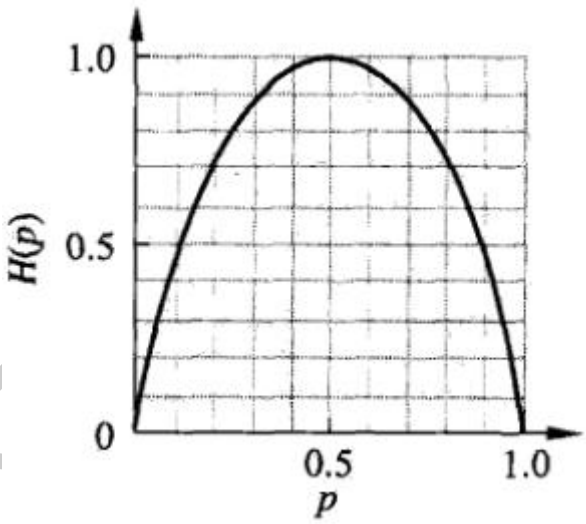
$$H(X) = -\sum_{i=1}^{n} p_i \log_2 p_i$$

当随机变量只取 2 个值 0 和 1 的时候，$p(X=1)=p$，$p(X=0)=1-p$，其中 $0 \leqslant p \leqslant 1$

熵为:

$$H(p) = -p\log_2 p - (1-p)\log_2(1-p)$$

熵随概率 P 的变化曲线如下图所示：



我们可以用求导来求解最大值的点：

$$H'(P) = \log_2 \frac{1-p}{p} + \frac{1}{\frac{1-p}{p}\ln 2} \times \frac{-1}{p} + \frac{1}{(1-p)\ln 2}$$

导函数为 0 时，我们求得:

$$\frac{1-p}{p} = 1$$

得到:$p = \frac{1}{2}$

1.2　信息增益（information gain)

表示得知特征 X 的信息而使得类 Y 的信息的不确定性减少的程度。

定理. 特征 A 对训练数据集 D 的信息增益 $g(D,A)$)，定义集合 D 的经验熵 $H(D)$ 与特征 A 给定条件下 D 的经验条件熵 $H(D|A)$ 之差，即:

$$g(D,A) = H(D) - H(D|A)$$

一般地，熵与条件熵之差成为互信息 mutual information ），决策树学习中的信息增益等价于训练数据集中类与特征的互信息。

定理. 互信息是两个随机变量间相互依赖性的量度，用 $I(X;Y)$ 表示。

互信息度量两个随机变量共享的信息——知道随机变量 X, 对随机变量 Y 的不确定性减少的程度（或者知道随机变量 Y，对随机变量 X 的不确定性减少的程度。

代入熵，条件熵的公式，经过计算与化简后：

$$I(X;Y)=\sum_{x,y}p(x,y)\log\frac{p(x,y)}{p(x)p(y)}$$

根据信息增益准则的特征选择方法是：对训练数据集（或子集）D，计算其每个特征的信息增益，并比较它们的大小，选择信息增益最大的特征。

1.3　信息增益计算算法

(1) 计算数据集 D 的经验熵

$$H(D)=-\sum_{k=1}^{K}\frac{|C_k|}{|D|}\log_2\frac{|C_k|}{|D|}$$

(2) 计算特征 A 对数据集 D 的经验条件熵 $H(D|A)$

$$H(D|A)=\sum_{i=1}^{n}\frac{|D_i|}{|D|}H(D_i)=-\sum_{i=1}^{n}\frac{|D_i|}{|D|}\sum_{k=1}^{K}\frac{|D_{ik}|}{|D_i|}\log_2\frac{|D_{ik}|}{|D_i|}$$

(3) 计算信息增益

$$g(D,A)=H(D)-H(D|A)$$

信息增益比的计算是:

$$g_R(D,A)=\frac{g(D,A)}{H(D)}$$

我们来看下面一个例子:

ID	年龄	有工作	有自己的房子	信贷情况	类别
1	青年	否	否	一般	否
2	青年	否	否	好	否
3	青年	是	否	好	是
4	青年	是	是	一般	是
5	青年	否	否	一般	否
6	中年	否	否	一般	否
7	中年	否	否	好	否
8	中年	是	是	好	是
9	中年	否	是	非常好	是
10	中年	否	是	非常好	是
11	老年	否	是	非常好	是
12	老年	否	是	好	是
13	老年	是	否	好	是
14	老年	是	否	非常好	是
15	老年	否	否	一般	否

首先我们计算熵为:

$$H(D) = -\frac{9}{15}\log_2\frac{9}{15} - \frac{6}{15}\log_2\frac{6}{15} = 0.971$$

解释：最后的类别中有 9 个" 是" 和 7 个" 否"，所以这里选择的是 9/15 和 6/15

以 A_1, A_2, A_3, A_4 表示年龄，有工作，有自己的房子，信贷情况 4 个特征，然后计算信息增益：

$$\begin{aligned} g(D, A_1) &= H(D) - [\frac{5}{15}H(D_1) + \frac{5}{15}H(D_2) + \frac{5}{15}H(D_3)] \\ &= 0.971 - [\frac{5}{15}(-\frac{2}{5}\log_2\frac{2}{5} - \frac{3}{5}\log_2\frac{3}{5}) + \frac{5}{15}(-\frac{2}{5}\log_2\frac{2}{5} - \frac{3}{5}\log_2\frac{3}{5}) \\ &+ \frac{5}{15}(-\frac{4}{5}\log_2\frac{4}{5} - \frac{1}{5}\log_2\frac{1}{5})] \\ &= 0.971 - 0.888 = 0.083 \end{aligned}$$

年龄中有 5 个青年，5 个中年，5 个老年，所以是 3 个 5/15，青年中有 2 个标签是“是”,3 个是“否”; 中年中是 3 个“是”，2 个“否”，老年中有 4 个“是”，1 个“否”。

依照这种思路计算可以得到:

$$\begin{aligned} g(D, A_2) &= H(D) - [\frac{5}{15}H(D_1) + \frac{10}{15}H(D_2)] \\ &= 0.971 - [\frac{5}{15} \times 0 + \frac{10}{15}(-\frac{4}{10}\log_2\frac{4}{10} - \frac{6}{10}\log_2\frac{6}{10})] \\ &= 0.324 \end{aligned}$$

$$
\begin{aligned}
g(D, A_3) &= 0.971 - [\frac{6}{15} \times 0 + \frac{9}{15}(-\frac{3}{9}\log_2\frac{3}{9} - \frac{6}{9}\log_2\frac{6}{9})] \\
&= 0.971 - 0.551 \\
&= 0.420
\end{aligned}
$$

1.4 数据集

使用数据集：周志华《机器学习》西瓜数据集。

我们的数据集展示为:

数据集展示为:

	色泽	根蒂	敲声	纹理	脐部	触感	好瓜
0	青绿	蜷缩	浊响	清晰	凹陷	硬滑	1
1	乌黑	蜷缩	沉闷	清晰	凹陷	硬滑	1
2	乌黑	蜷缩	浊响	清晰	凹陷	硬滑	1
3	青绿	蜷缩	沉闷	清晰	凹陷	硬滑	1
4	浅白	蜷缩	浊响	清晰	凹陷	硬滑	1
5	青绿	稍蜷	浊响	清晰	稍凹	软粘	1
6	乌黑	稍蜷	浊响	稍糊	稍凹	软粘	1
7	乌黑	稍蜷	浊响	清晰	稍凹	硬滑	1
8	乌黑	稍蜷	沉闷	稍糊	稍凹	硬滑	0
9	青绿	硬挺	清脆	清晰	平坦	软粘	0
10	浅白	硬挺	清脆	模糊	平坦	硬滑	0
11	浅白	蜷缩	浊响	模糊	平坦	软粘	0
12	青绿	稍蜷	浊响	稍糊	凹陷	硬滑	0
13	浅白	稍蜷	沉闷	稍糊	凹陷	硬滑	0
14	乌黑	稍蜷	浊响	清晰	稍凹	软粘	0
15	浅白	蜷缩	浊响	模糊	平坦	硬滑	0
16	青绿	蜷缩	沉闷	稍糊	稍凹	硬滑	0

1.5 ID3 决策树基础代码实现

ID3 算法的核心是在决策树各个结点上对应信息增益准则选择特征，递归地构建决策树。具体方法是：从根结点 (root node) 开始，对结点计算所有可能的特征的信息增益，选择信息增益最大的特征作为结点的特征，由该特征的不同取值建立子节点；再对子结点递归地调用以上方法，构建决策树；直到所有特征的信息增益均很小或没有特征可以选择为止，最后得到一个决策树。ID3 相当于用极大似然法进行概率模型的选择。

根据决策树算法，可知 ID3 决策树的算法流程如下：

1. 先根据最大信息增益选取一个特征作为根节点

2. 以根节点特征的取值作为分支递归生成节点，在递归中注意：

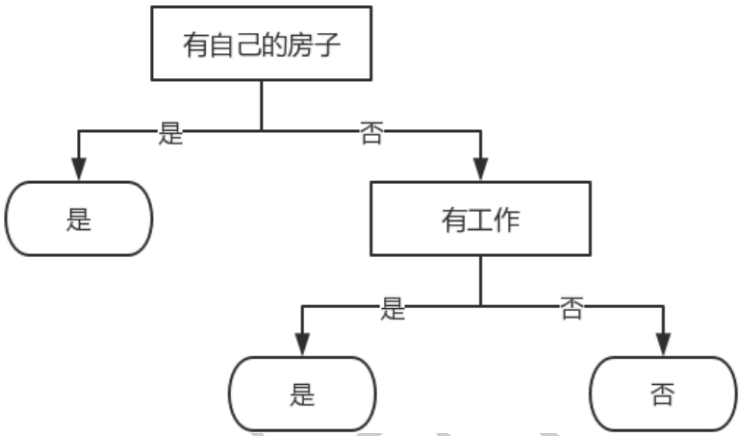
(1) 每次取特征值时需要删除之前取过的数据

(2) 当当前样本只有一类时，返回该类别作叶子结点，即分类结果

(3) 当当前所有样本的特征值都一样时，选样本最多的类作为叶子结点

3. 使用测试特征测试决策树预测能力

我们使用字典存储决策树的结构，比如下图分析出来的决策树，用字典可以表示为：




```
{'有自己的房子': {0: {'有工作': {0: 'no', 1: 'yes'}}, 1: 'yes'}}
```

1.5.1 计算信息熵

首先我们计算给定数据的信息熵:

```
def cal_information_entropy(data):
    data_labels=data.iloc[:,-1]
    label_class=data_labels.value_counts() #标签计数
    entropy=0.0
    for key in label_class.keys():
        prob=label_class[key]/len(data_labels)
        entropy+=-prob*log(prob,2)
    return entropy,label_class
entropy,label_calss=cal_information_entropy(data)
print('entropy=\n',entropy)
```

我们得出的计算结果为:

```
entropy=
 0.9975025463691153
```

我们不妨来验证一下，对于上面的数据集，其中包含 17 个训练样例。显然，标签的类型为两种，1 和 0，其中 1 代表好瓜，0 代表坏瓜。根据所有样例，其中正例占 $p_1 = \frac{8}{17}$，反例占 $p_2 = \frac{9}{17}$。于是，根据上面的计算公式，我们可以计算出根结点的信息熵为:

$$\begin{aligned} Ent(D) &= -\sum_{k=1}^{2} p_k \log_2 p_k \\ &= -(\frac{8}{17}\log_2\frac{8}{17} + \frac{9}{17}\log_2\frac{9}{17}) \\ &= 0.998 \end{aligned}$$

1.5.2 信息增益

然后，我们要计算当前属性集合 {色泽，根蒂，敲声，纹理，脐部，触感} 中每个属性的信息增益。以属性“色泽”为例，它有三个可能的取值:{青绿，乌黑，浅白}。若使用该属性对数据集 D 进行划分，可以得到三个子集，分别记为:D_1(色泽 = 青绿),D_2(色泽 = 乌黑)，D_3(色泽 = 浅白)。

三个不同的取值的个数分别为:

```
青绿    6
乌黑    6
浅白    5
```

对于子集 D_1，其中正例占 $p_1 = \frac{3}{6}$，反例占 $p_2 = \frac{3}{6}$

对于子集 D_2，其中正例占 $p_1 = \frac{4}{6}$，反例占 $p_2 = \frac{2}{6}$

对于子集 D_3，其中正例占 $p_1 = \frac{1}{5}$，反例占 $p_2 = \frac{4}{5}$

根据公式可计算出“色泽”划分之后所获得的 3 个分支结点的信息熵为:

$$\begin{aligned} Ent(D_1) &= -(\frac{3}{6}\log_2\frac{3}{6} + \frac{3}{6}\log_2\frac{3}{6}) \\ &= 1.00 \end{aligned}$$

$$\begin{aligned} Ent(D_2) &= -(\frac{4}{6}\log_2\frac{4}{6} + \frac{2}{6}\log_2\frac{2}{6}) \\ &= 0.918 \end{aligned}$$

$$\begin{aligned} Ent(D_1) &= -(\frac{1}{5}\log_2\frac{1}{5} + \frac{4}{5}\log_2\frac{4}{5}) \\ &= 0.722 \end{aligned}$$

于是，属性 d_1=“色泽”的信息增益为:

$$\begin{aligned} Gain(D, d_1) &= Ent(D) - \sum_{v=1}^{3} \frac{|D_v|}{|D|} Ent(D_v) \\ &= 0.998 - (\frac{6}{17} \times 1.000 + \frac{6}{17} \times 0.918 + \frac{5}{17} \times 0.722) \\ &= 0.109 \end{aligned}$$

类似的，我们可以计算出其他属性的信息增益:

$$Gain(D, d_2) = 0.143; Gain(D, d_3) = 0.141$$

$$Gain(D, d_4) = 0.381; Gain(D, d_5) = 0.289$$

$$Gain(D, d_6) = 0.006$$

我们用 python 实现如下:

```
#计算给定数据属性a的信息增益
def cal_information_gain(data,a):
    entropy = cal_information_entropy(data)
    feature_class = data[a].value_counts() #特征有多少种可能
    gain = 0
    for v in feature_class.keys():
        weight = feature_class[v]/data.shape[0]
        entropy_v = cal_information_entropy(data.loc[data[a] == v])[0]
        gain+=weight*entropy_v
    info_gain=entropy-gain
    return info_gain
print('=='*30)
print('色泽的信息增益:',cal_information_gain(data,'色泽'))
print('=='*30)
print('根蒂的信息增益:',cal_information_gain(data,'根蒂'))
print('=='*30)
print('敲声的信息增益:',cal_information_gain(data,'敲声'))
print('=='*30)
print('脐部的信息增益:',cal_information_gain(data,'脐部'))
print('=='*30)
print('触感的信息增益:',cal_information_gain(data,'触感'))
print('=='*30)
```

得到的结果如下:

```
============================================================
```

```
色泽的信息增益: [0.10812516526536531 0    8.110623
                 1    7.110623
                 Name: 好瓜, dtype: float64]
============================================================
根蒂的信息增益: [0.14267495956679288 0    8.145172
                 1    7.145172
                 Name: 好瓜, dtype: float64]
============================================================
敲声的信息增益: [0.14078143361499584 0    8.143279
                 1    7.143279
                 Name: 好瓜, dtype: float64]
============================================================
脐部的信息增益: [0.28915878284167895 0    8.291656
                 1    7.291656
                 Name: 好瓜, dtype: float64]
============================================================
触感的信息增益: [0.006046489176565584 0    8.008544
                  1    7.008544
                  Name: 好瓜, dtype: float64]
============================================================
```

从上面的输出结果可以看出，我们的数据类型为数组的形式，不符合我们的输出要求，因此我们做一下改动，代码修改如下:

```
def cal_information_entropy(data):
    data_labels=data.iloc[:,-1]
    label_class=data_labels.value_counts() #标签计数
    entropy=0
    for key in label_class.keys():
        prob=label_class[key]/len(data_labels)
        entropy+=-prob*log(prob,2)
    return entropy
entropy=cal_information_entropy(data)
print('=='*30)
print('entropy=\n',entropy)
print('=='*30)
#计算给定数据属性a的信息增益
def cal_information_gain(data,a):
    entropy = cal_information_entropy(data)
    feature_class = data[a].value_counts()#特征有多少种可能
```

```
    print(feature_class)
    print('特征类型:\n', feature_class.keys())
    print('==' * 30)
    gain_entropy = 0.0
    for v in feature_class.keys():
        weight = feature_class[v]/data.shape[0]
        entropy_v= cal_information_entropy(data.loc[data[a] == v])
        condition_entropy=weight*entropy_v #条件熵的每一项
        gain_entropy+=weight*entropy_v
        print('权重值为:\n',weight)
        print('对应的熵值为:\n',entropy_v)
        print('对应的条件熵的子集为:\n',condition_entropy)
        print(data.loc[data[a] == v],type(data.loc[data[a] == v]))
        print('=='*30)
    infomation_entropy_gain = entropy - gain_entropy
    return infomation_entropy_gain
print('色泽的信息增益为:\n',cal_information_gain(data,'色泽'))
print('==' * 30)
print('根蒂的信息增益为:\n',cal_information_gain(data,'根蒂'))
print('==' * 30)
print('敲声的信息增益为:\n',cal_information_gain(data,'敲声'))
print('==' * 30)
print('纹理的信息增益为:\n',cal_information_gain(data,'纹理'))
print('==' * 30)
print('脐部的信息增益为:\n',cal_information_gain(data,'脐部'))
print('==' * 30)
print('触感的信息增益为:\n',cal_information_gain(data,'触感'))
print('==' * 30)
```

我们得出的结果如下:

```
============================================================
entropy=
 0.9975025463691153
============================================================
青绿    6
乌黑    6
浅白    5
Name: 色泽, dtype: int64
特征类型:
```

```
 Index(['青绿', '乌黑', '浅白'], dtype='object')
==============================================================
权重值为:
 0.35294117647058826
对应的熵值为:
 1.0
对应的条件熵的子集为:
 0.35294117647058826
     色泽   根蒂   敲声   纹理   脐部   触感   好瓜
0    青绿   蜷缩   浊响   清晰   凹陷   硬滑    1
3    青绿   蜷缩   沉闷   清晰   凹陷   硬滑    1
5    青绿   稍蜷   浊响   清晰   稍凹   软粘    1
9    青绿   硬挺   清脆   清晰   平坦   软粘    0
12   青绿   稍蜷   浊响   稍糊   凹陷   硬滑    0
16   青绿   蜷缩   沉闷   稍糊   稍凹   硬滑    0 <class 'pandas.core.frame.
                                                DataFrame'>
==============================================================
权重值为:
 0.35294117647058826
对应的熵值为:
 0.9182958340544896
对应的条件熵的子集为:
 0.3241044120192316
     色泽   根蒂   敲声   纹理   脐部   触感   好瓜
1    乌黑   蜷缩   沉闷   清晰   凹陷   硬滑    1
2    乌黑   蜷缩   浊响   清晰   凹陷   硬滑    1
6    乌黑   稍蜷   浊响   稍糊   稍凹   软粘    1
7    乌黑   稍蜷   浊响   清晰   稍凹   硬滑    1
8    乌黑   稍蜷   沉闷   稍糊   稍凹   硬滑    0
14   乌黑   稍蜷   浊响   清晰   稍凹   软粘    0 <class 'pandas.core.frame.
                                                DataFrame'>
==============================================================
权重值为:
 0.29411764705882354
对应的熵值为:
 0.7219280948873623
对应的条件熵的子集为:
 0.2123317926139301
     色泽   根蒂   敲声   纹理   脐部   触感   好瓜
```

```
4   浅白  蜷缩  浊响  清晰  凹陷  硬滑   1
10  浅白  硬挺  清脆  模糊  平坦  硬滑   0
11  浅白  蜷缩  浊响  模糊  平坦  软粘   0
13  浅白  稍蜷  沉闷  稍糊  凹陷  硬滑   0
15  浅白  蜷缩  浊响  模糊  平坦  硬滑   0 <class 'pandas.core.frame.
                                        DataFrame'>
============================================================
色泽的信息增益为:
 0.10812516526536531
============================================================
蜷缩    8
稍蜷    7
硬挺    2
Name: 根蒂, dtype: int64
特征类型:
 Index(['蜷缩', '稍蜷', '硬挺'], dtype='object')
============================================================
权重值为:
 0.47058823529411764
对应的熵值为:
 0.9544340029249649
对应的条件熵的子集为:
 0.44914541314115997
    色泽  根蒂  敲声  纹理  脐部  触感  好瓜
0   青绿  蜷缩  浊响  清晰  凹陷  硬滑   1
1   乌黑  蜷缩  沉闷  清晰  凹陷  硬滑   1
2   乌黑  蜷缩  浊响  清晰  凹陷  硬滑   1
3   青绿  蜷缩  沉闷  清晰  凹陷  硬滑   1
4   浅白  蜷缩  浊响  清晰  凹陷  硬滑   1
11  浅白  蜷缩  浊响  模糊  平坦  软粘   0
15  浅白  蜷缩  浊响  模糊  平坦  硬滑   0
16  青绿  蜷缩  沉闷  稍糊  稍凹  硬滑   0 <class 'pandas.core.frame.
                                        DataFrame'>
============================================================
权重值为:
 0.4117647058823529
对应的熵值为:
 0.9852281360342516
对应的条件熵的子集为:
```

```
 0.40568217366116244
    色泽  根蒂  敲声  纹理  脐部  触感  好瓜
5   青绿  稍蜷  浊响  清晰  稍凹  软粘   1
6   乌黑  稍蜷  浊响  稍糊  稍凹  软粘   1
7   乌黑  稍蜷  浊响  清晰  稍凹  硬滑   1
8   乌黑  稍蜷  沉闷  稍糊  稍凹  硬滑   0
12  青绿  稍蜷  浊响  稍糊  凹陷  硬滑   0
13  浅白  稍蜷  沉闷  稍糊  凹陷  硬滑   0
14  乌黑  稍蜷  浊响  清晰  稍凹  软粘   0 <class 'pandas.core.frame.
                                          DataFrame'>
============================================================
权重值为:
 0.11764705882352941
对应的熵值为:
 0.0
对应的条件熵的子集为:
 0.0
    色泽  根蒂  敲声  纹理  脐部  触感  好瓜
9   青绿  硬挺  清脆  清晰  平坦  软粘   0
10  浅白  硬挺  清脆  模糊  平坦  硬滑   0 <class 'pandas.core.frame.
                                          DataFrame'>
============================================================
根蒂的信息增益为:
 0.14267495956679288
============================================================
浊响    10
沉闷     5
清脆     2
Name: 敲声, dtype: int64
特征类型:
 Index(['浊响', '沉闷', '清脆'], dtype='object')
============================================================
权重值为:
 0.5882352941176471
对应的熵值为:
 0.9709505944546686
对应的条件熵的子集为:
 0.5711474085027463
    色泽  根蒂  敲声  纹理  脐部  触感  好瓜
```

```
0    青绿  蜷缩  浊响  清晰  凹陷  硬滑    1
2    乌黑  蜷缩  浊响  清晰  凹陷  硬滑    1
4    浅白  蜷缩  浊响  清晰  凹陷  硬滑    1
5    青绿  稍蜷  浊响  清晰  稍凹  软粘    1
6    乌黑  稍蜷  浊响  稍糊  稍凹  软粘    1
7    乌黑  稍蜷  浊响  清晰  稍凹  硬滑    1
11   浅白  蜷缩  浊响  模糊  平坦  软粘    0
12   青绿  稍蜷  浊响  稍糊  凹陷  硬滑    0
14   乌黑  稍蜷  浊响  清晰  稍凹  软粘    0
15   浅白  蜷缩  浊响  模糊  平坦  硬滑    0 <class 'pandas.core.frame.
                                          DataFrame'>
============================================================
权重值为:
 0.29411764705882354
对应的熵值为:
 0.9709505944546686
对应的条件熵的子集为:
 0.28557370425137313
     色泽  根蒂  敲声  纹理  脐部  触感  好瓜
1    乌黑  蜷缩  沉闷  清晰  凹陷  硬滑    1
3    青绿  蜷缩  沉闷  清晰  凹陷  硬滑    1
8    乌黑  稍蜷  沉闷  稍糊  稍凹  硬滑    0
13   浅白  稍蜷  沉闷  稍糊  凹陷  硬滑    0
16   青绿  蜷缩  沉闷  稍糊  稍凹  硬滑    0 <class 'pandas.core.frame.
                                          DataFrame'>
============================================================
权重值为:
 0.11764705882352941
对应的熵值为:
 0.0
对应的条件熵的子集为:
 0.0
     色泽  根蒂  敲声  纹理  脐部  触感  好瓜
9    青绿  硬挺  清脆  清晰  平坦  软粘    0
10   浅白  硬挺  清脆  模糊  平坦  硬滑    0 <class 'pandas.core.frame.
                                          DataFrame'>
============================================================
敲声的信息增益为:
 0.14078143361499584
```

```
============================================================
清晰    9
稍糊    5
模糊    3
Name: 纹理, dtype: int64
特征类型:
 Index(['清晰', '稍糊', '模糊'], dtype='object')
============================================================
权重值为:
 0.5294117647058824
对应的熵值为:
 0.7642045065086203
对应的条件熵的子集为:
 0.4045788563869166
     色泽   根蒂   敲声   纹理   脐部   触感   好瓜
0    青绿   蜷缩   浊响   清晰   凹陷   硬滑    1
1    乌黑   蜷缩   沉闷   清晰   凹陷   硬滑    1
2    乌黑   蜷缩   浊响   清晰   凹陷   硬滑    1
3    青绿   蜷缩   沉闷   清晰   凹陷   硬滑    1
4    浅白   蜷缩   浊响   清晰   凹陷   硬滑    1
5    青绿   稍蜷   浊响   清晰   稍凹   软粘    1
7    乌黑   稍蜷   浊响   清晰   稍凹   硬滑    1
9    青绿   硬挺   清脆   清晰   平坦   软粘    0
14   乌黑   稍蜷   浊响   清晰   稍凹   软粘    0 <class 'pandas.core.frame.
                                                  DataFrame'>
============================================================
权重值为:
 0.29411764705882354
对应的熵值为:
 0.7219280948873623
对应的条件熵的子集为:
 0.2123317926139301
     色泽   根蒂   敲声   纹理   脐部   触感   好瓜
6    乌黑   稍蜷   浊响   稍糊   稍凹   软粘    1
8    乌黑   稍蜷   沉闷   稍糊   稍凹   硬滑    0
12   青绿   稍蜷   浊响   稍糊   凹陷   硬滑    0
13   浅白   稍蜷   沉闷   稍糊   凹陷   硬滑    0
16   青绿   蜷缩   沉闷   稍糊   稍凹   硬滑    0 <class 'pandas.core.frame.
                                                  DataFrame'>
```

```
============================================================
权重值为:
 0.17647058823529413
对应的熵值为:
 0.0
对应的条件熵的子集为:
 0.0
     色泽   根蒂   敲声   纹理   脐部   触感   好瓜
10   浅白   硬挺   清脆   模糊   平坦   硬滑    0
11   浅白   蜷缩   浊响   模糊   平坦   软粘    0
15   浅白   蜷缩   浊响   模糊   平坦   硬滑    0 <class 'pandas.core.frame.
                                          DataFrame'>
============================================================
纹理的信息增益为:
 0.3805918973682686
============================================================
凹陷     7
稍凹     6
平坦     4
Name: 脐部, dtype: int64
特征类型:
 Index(['凹陷', '稍凹', '平坦'], dtype='object')
============================================================
权重值为:
 0.4117647058823529
对应的熵值为:
 0.863120568566631
对应的条件熵的子集为:
 0.355402587056848
     色泽   根蒂   敲声   纹理   脐部   触感   好瓜
0    青绿   蜷缩   浊响   清晰   凹陷   硬滑    1
1    乌黑   蜷缩   沉闷   清晰   凹陷   硬滑    1
2    乌黑   蜷缩   浊响   清晰   凹陷   硬滑    1
3    青绿   蜷缩   沉闷   清晰   凹陷   硬滑    1
4    浅白   蜷缩   浊响   清晰   凹陷   硬滑    1
12   青绿   稍蜷   浊响   稍糊   凹陷   硬滑    0
13   浅白   稍蜷   沉闷   稍糊   凹陷   硬滑    0 <class 'pandas.core.frame.
                                          DataFrame'>
============================================================
```

```
权重值为:
 0.35294117647058826
对应的熵值为:
 1.0
对应的条件熵的子集为:
 0.35294117647058826
     色泽   根蒂   敲声   纹理   脐部   触感   好瓜
5    青绿   稍蜷   浊响   清晰   稍凹   软粘    1
6    乌黑   稍蜷   浊响   稍糊   稍凹   软粘    1
7    乌黑   稍蜷   浊响   清晰   稍凹   硬滑    1
8    乌黑   稍蜷   沉闷   稍糊   稍凹   硬滑    0
14   乌黑   稍蜷   浊响   清晰   稍凹   软粘    0
16   青绿   蜷缩   沉闷   稍糊   稍凹   硬滑    0 <class 'pandas.core.frame.
                                                 DataFrame'>
============================================================
权重值为:
 0.23529411764705882
对应的熵值为:
 0.0
对应的条件熵的子集为:
 0.0
     色泽   根蒂   敲声   纹理   脐部   触感   好瓜
9    青绿   硬挺   清脆   清晰   平坦   软粘    0
10   浅白   硬挺   清脆   模糊   平坦   硬滑    0
11   浅白   蜷缩   浊响   模糊   平坦   软粘    0
15   浅白   蜷缩   浊响   模糊   平坦   硬滑    0 <class 'pandas.core.frame.
                                                 DataFrame'>
============================================================
脐部的信息增益为:
 0.28915878284167895
============================================================
硬滑     12
软粘      5
Name: 触感, dtype: int64
特征类型:
 Index(['硬滑', '软粘'], dtype='object')
============================================================
权重值为:
 0.7058823529411765
```

```
对应的熵值为:
 1.0
对应的条件熵的子集为:
 0.7058823529411765
    色泽  根蒂  敲声  纹理  脐部  触感  好瓜
0   青绿  蜷缩  浊响  清晰  凹陷  硬滑   1
1   乌黑  蜷缩  沉闷  清晰  凹陷  硬滑   1
2   乌黑  蜷缩  浊响  清晰  凹陷  硬滑   1
3   青绿  蜷缩  沉闷  清晰  凹陷  硬滑   1
4   浅白  蜷缩  浊响  清晰  凹陷  硬滑   1
7   乌黑  稍蜷  浊响  清晰  稍凹  硬滑   1
8   乌黑  稍蜷  沉闷  稍糊  稍凹  硬滑   0
10  浅白  硬挺  清脆  模糊  平坦  硬滑   0
12  青绿  稍蜷  浊响  稍糊  凹陷  硬滑   0
13  浅白  稍蜷  沉闷  稍糊  凹陷  硬滑   0
15  浅白  蜷缩  浊响  模糊  平坦  硬滑   0
16  青绿  蜷缩  沉闷  稍糊  稍凹  硬滑   0 <class 'pandas.core.frame.
                                          DataFrame'>
============================================================
权重值为:
 0.29411764705882354
对应的熵值为:
 0.9709505944546686
对应的条件熵的子集为:
 0.28557370425137313
    色泽  根蒂  敲声  纹理  脐部  触感  好瓜
5   青绿  稍蜷  浊响  清晰  稍凹  软粘   1
6   乌黑  稍蜷  浊响  稍糊  稍凹  软粘   1
9   青绿  硬挺  清脆  清晰  平坦  软粘   0
11  浅白  蜷缩  浊响  模糊  平坦  软粘   0
14  乌黑  稍蜷  浊响  清晰  稍凹  软粘   0 <class 'pandas.core.frame.
                                          DataFrame'>
============================================================
触感的信息增益为:
 0.006046489176565584
============================================================
```

我们接下来获取标签最多的那一类:

```
def get_most_label(data):
```

```
    data_label = data.iloc[:,-1]
    label_sort = data_label.value_counts(sort=True)
    return label_sort.keys()[0]
print('获取标签最多的一类为:\n',get_most_label(data))
```

结果展示为:

```
获取标签最多的一类为:
 0
```

接下来我们来挑选最优特征，即信息增益最大的特征，代码实现如下:

```
def get_best_feature(data):
    features = data.columns[:-1]
    res = {}
    for a in features:
        info_gain = cal_information_gain(data,a)
        res[a] = info_gain
    res = sorted(res.items(),key=lambda x:x[1],reverse=True)
    return res[0][0]
print('获取分类的最佳特征:\n',get_best_feature(data))
```

最后的输出结果如下所示;

```
获取分类的最佳特征:
 纹理
```

通过上面的信息增益值我们可以看出，“纹理”的信息增益值最大，于是它被选为划分属性

将数据转化为（属性值：数据）的元组形式返回，并删除之前的特征列。

```
def drop_exist_feature(data, best_feature):
    attribute = pd.unique(data[best_feature])
    new_data = [(nd, data[data[best_feature] == nd]) for nd in attribute]
    new_data = [(n[0], n[1].drop([best_feature], axis=1)) for n in new_data
                                                        ]
    return new_data
```

我们展示一下每个特征下面的属性:

```
column_count = dict([(ds, list(pd.unique(data[ds]))) for ds in data.iloc[:,
                                                :-1].columns])
print(column_count)
```

```
{'色泽': ['青绿', '乌黑', '浅白'], '根蒂': ['蜷缩', '稍蜷', '硬挺'], '敲声'
                                    : ['浊响', '沉闷', '清脆'], '纹理': [
                                    '清晰', '稍糊', '模糊'], '脐部': ['凹
                                    陷', '稍凹', '平坦'], '触感': ['硬滑'
                                    , '软粘']}
```

决策树绘制基本参考《机器学习实战》书内的代码。

```
"""
函数说明:创建决策树
Parameters:
    dataSet - 训练数据集
    labels - 分类属性标签
    featLabels - 存储选择的最优特征标签
Returns:
    myTree - 决策树
"""
def create_tree(data):
    data_label = data.iloc[:,-1] #取分类标签
    if len(data_label.value_counts()) == 1: #只有一类
        return data_label.values[0]
    if all(len(data[i].value_counts()) == 1 for i in data.iloc[:,:-1].
                                        columns): #所有数据的特征值一样,
                                        选样本最多的类作为分类结果
        return get_most_label(data)
    best_feature = get_best_feature(data) #根据信息增益得到的最优划分特征
    Tree = {best_feature:{}} #用字典形式存储决策树
    exist_vals = pd.unique(data[best_feature]) #当前数据下最佳特征的取值
    if len(exist_vals) != len(column_count[best_feature]): #如果特征的取值
                                        相比于原来的少了
        no_exist_attr = set(column_count[best_feature]) - set(exist_vals) #
                                            少的那些特征
        for no_feat in no_exist_attr:
            Tree[best_feature][no_feat] = get_most_label(data) #缺失的特征
                                                分类为当前类别最多的


    for item in drop_exist_feature(data,best_feature): #根据特征值的不同递
                                        归创建决策树
        Tree[best_feature][item[0]] = create_tree(item[1])
```

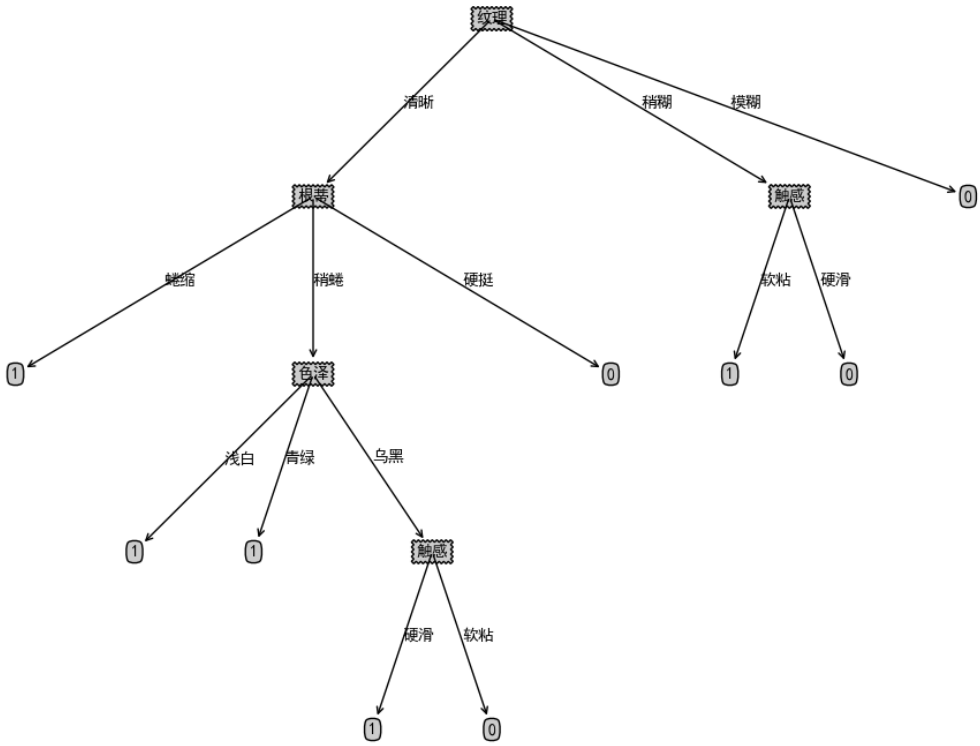
```
    return Tree


my_tree=create_tree(data)
print('构造的决策树如下:\n',my_tree)
```

此处我们生成的决策树结果为:

```
构造的决策树如下:
 {'纹理': {'清晰': {'根蒂': {'蜷缩': 1, '稍蜷': {'色泽': {'浅白': 1, '青绿'
                                                    : 1, '乌黑': {'触感': {'硬滑': 1, '
                                                    软粘': 0}}}}, '硬挺': 0}}, '稍糊': {
                                                    '触感': {'软粘': 1, '硬滑': 0}}, '模
                                                    糊': 0}}
```

可视化决策树的代码与上一周的学习笔记代码相同，此处不再赘述。可视化后的决策树如下所示:



最后我们可以建立一个预测模型:

```
def predict(Tree , test_data):
    first_feature = list(Tree.keys())[0]
```

```
    second_dict = Tree[first_feature]
    input_first = test_data.get(first_feature)
    input_value = second_dict[input_first]
    if isinstance(input_value , dict): #判断分支还是不是字典
        class_label = predict(input_value, test_data)
    else:
        class_label = input_value
    return class_label
test_data = {'色泽':'青绿','根蒂':'蜷缩','敲声':'浊响','纹理':'稍糊','脐部'
                                        :'凹陷','触感':'硬滑'}
print('预测结果为:\n',predict(my_tree,test_data))
```

最后的预测结果为 0，即对应的坏瓜。

```
预测结果为:
 0
```

第二章　C4.5 决策树

2.1　信息增益率

在上面的介绍中，我们有意忽略了表中的标号这一列，可以对应为 ID。若是把这个特征也作为一个候选划分属性，则根据前式可计算出它的信息增益为 0.998，远大于其他候选划分属性。这很容易理解:“ID”将产生 17 个分支，每个分支结点仅包含一个样本，这些分支结点的纯度已经达到最大。然而，这样的决策树显然不具有泛化能力，无法对新样本进行有效预测。

实际上，信息增益准则对可取值数目较多的属性有所偏好，为减少这种偏好可能带来的不利影响，著名的 C4.5 决策树算法不直接使用信息增益，而是使用“增益率”(gain ratio) 来选择最优划分属性，采用用上式相同的符号表示，增益率定义为:

$$Gain-ratio(D,a)=\frac{Gain(D,a)}{IV(a)}$$

其中:

$$IV(a)=-\sum_{v=1}^{V}\frac{|D_v|}{|D|}\log_2\frac{|D_v|}{|D|}$$

$IV(a)$ 称为属性 a 的“固有值”(intrinsic value)。属性 a 的可能数目越多（即 V 越大），则 $IV(a)$ 的值通常会越大。例如，对于上面的数据集，有 IV（触感）=0.874（$V=2$），IV（色泽）=1.580($V=3$)，

需要注意的是，增益率准则对可取值数目较少的属性有所偏好，因此，C4.5 算法并不是直接选择增益率最大的候选划分属性，而是使用了一个启发式：先从候选划分属性中找出信息增益高于平均水平的属性，再从中选择增益率最高的。

2.2　Python 代码实现

2.2.1　计算固有值和增益率

我们的代码实现如下:

```
def cal_gain_ratio(data , a):
    #先计算固有值intrinsic_value
    IV_a = 0
    feature_class = data[a].value_counts()  # 特征有多少种可能
    for v in feature_class.keys():
        weight = feature_class[v]/data.shape[0]
        IV_a += -weight*log(weight,2)
    gain_ration = cal_information_gain(data,a)/IV_a
    return IV_a,gain_ration

for a in ['色泽','根蒂','敲声','纹理','脐部','触感']:
    IV_a,gain_ration=cal_gain_ratio(data,a)
    print('属性为%s'%a)
    print(a,'属性的固有值:\n',IV_a)
    print(a,'属性的增益率为:\n',gain_ration)
    print('=='*30)
```

我们的输出结果为:

```
============================================================
属性为色泽
色泽 属性的固有值:
 1.5798634010685344
色泽 属性的增益率为:
 0.06843956584615814
============================================================
属性为根蒂
根蒂 属性的固有值:
 1.402081402756032
根蒂 属性的增益率为:
 0.1017593980537369
============================================================
属性为敲声
敲声 属性的固有值:
 1.3328204045850196
敲声 属性的增益率为:
 0.10562670944314426
============================================================
属性为纹理
```

```
纹理 属性的固有值:
 1.4466479595102755
纹理 属性的增益率为:
 0.2630853587192754
============================================================
属性为脐部
脐部 属性的固有值:
 1.548565226030918
脐部 属性的增益率为:
 0.1867268991844879
============================================================
属性为触感
触感 属性的固有值:
 0.8739810481273578
触感 属性的增益率为:
 0.0069183298534003
============================================================
```

获取最优划分特征时，与 ID3 算法不同，我们先从候选划分属性中找出信息增益高于平均水平的属性，再从中选择增益率最高的。

```
def get_best_feature(data):
    features = data.columns[:-1]
    res = {}
    for a in features:
        temp = cal_information_gain(data, a)
        gain_ration = cal_gain_ratio(data,a)
        res[a] = (temp,gain_ration)
    res = sorted(res.items(),key=lambda x:x[1][0],reverse=True) #按信息增益
                                                排名
    res_avg = sum([x[1][0] for x in res])/len(res) #信息增益平均水平
    good_res = [x for x in res if x[1][0] >= res_avg] #选取信息增益高于平均
                                                水平的特征
    result =sorted(good_res,key=lambda x:x[1][1],reverse=True) #将信息增益
                                                高的特征按照增益率进行排名
    return result[0][0] #返回高信息增益中增益率最大的特征
print('获取最佳的划分属性:\n',get_best_feature(data))
```

最终的展示结果为:

```
获取最佳的划分属性:
 纹理
```

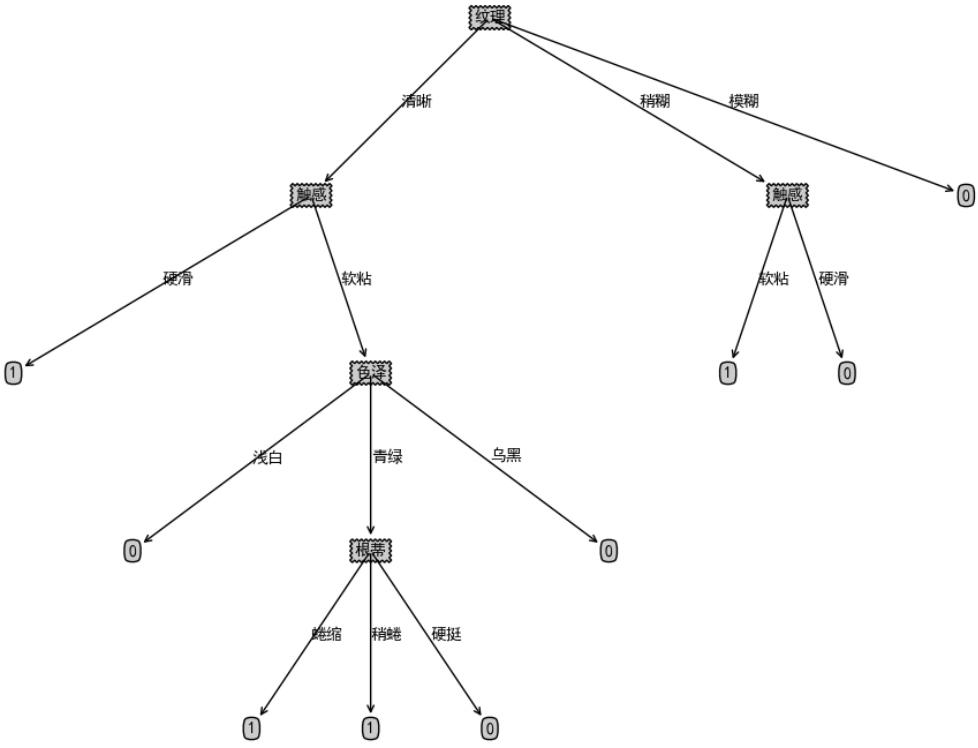
接下来的步骤与 ID3 算法一致，这里不再赘述。

这里展示输出结果为:

```
获取标签最多的一类为:
 0
```

```
构造的决策树如下:
{'纹理': {'清晰': {'触感': {'硬滑': 1, '软粘': {'色泽': {'浅白': 0, '青绿'
                                                    : {'根蒂': {'蜷缩': 1, '稍蜷': 1, '
                                                    硬挺': 0}}, '乌黑': 0}}}}, '稍糊': {
                                                    '触感': {'软粘': 1, '硬滑': 0}}, '模
                                                    糊': 0}}
```

可视化的决策树如下所示:



以上就是 C4.5 算法的基本实现。

第三章　CART 决策树

CART 决策树与前面两个都不同，它采用基尼指数划分属性，计算公式如下：

$$\begin{aligned} Gini(D) &= \sum_{k=1}^{|y|} p_k(1-p_k) \\ &= 1 - \sum_{k=1}^{|y|} p_k^2 \end{aligned}$$

其中，$|y|$ 表示类别个数。当做二分类时，公式可以简化为：

$$Gini(D) = 2p(1-p)$$

从公式上来理解，基尼指数表示了在样本中随机抽两个样本，其类别不一样的概率，值越小说明一个类别明显多于另一个类别，纯度越高。

举个例子：当二分类样本对半分的时候，基尼指数为 0.5，而样本 46 开的时候，基尼指数为 0.48

当我们划分属性时，需要对基尼指数赋予不同的权重（与前面的信息增益样本权重一致），公式为：

$$Gini - index(D,a) = \sum_{v=1}^{V} \frac{|D_v|}{|D|} Gini(D_v)$$

如果样本集合根据特征 A 是否取某一可能值 a 被分割成 D_1 和 D_2 两部分，即:

$$D_1 = \{(x,y) \in D | A(x) = a\}, D_2 = D - D_1$$

则在特征 A 的条件下，集合 D 的基尼指数定义为:

$$Gini(D,A) = \frac{|D_1|}{|D|} Gini(D_1) + \frac{|D_2|}{|D|} Gini(D_2)$$

在划分属性时，选择划分后基尼指数最小的属性作为最优划分属性。

CART 生成算法的步骤如下:

输入：训练数据集 D，停止计算的条件；

输出：CART 决策树

根据训练数据集，从根结点开始，递归的对每个结点进行以下操作，构建二叉决策树。

（1）设结点的训练数据集为 D，计算现有特征对该数据集的基尼系数。此时，对每一个特征 A，对其可能取的每个值 a，根据样本点对 $A=a$ 的测试为“是”或者“否”，将 D 分割成 D_1 和 D_2 两部分，利用上式计算 $A=a$ 时的基尼系数。

（2）在所有可能的特征 A 以及它们所有可能的切分点 a 中，选择基尼系数最小的特征及其对应的分类点作为最优特征与最优切分点。依最优特征与最优切分点，从现结点生成两个子结点，将训练数据集依特征分配到两个子结点中去。

（3）对两个子结点递归的调用（1）（2），直到满足停止条件。

（4）生成 CART 决策树。

我们继续采用上面的数据集来进行演示，数据集如下:

ID	年龄	有工作	有自己的房子	信贷情况	类别
1	青年	否	否	一般	否
2	青年	否	否	好	否
3	青年	是	否	好	是
4	青年	是	是	一般	是
5	青年	否	否	一般	否
6	中年	否	否	一般	否
7	中年	否	否	好	否
8	中年	是	是	好	是
9	中年	否	是	非常好	是
10	中年	否	是	非常好	是
11	老年	否	是	非常好	是
12	老年	否	是	好	是
13	老年	是	否	好	是
14	老年	是	否	非常好	是
15	老年	否	否	一般	否

首先计算各特征的基尼系数，选择最优特征以及最优切分点。仍采用上例的记号，分别以 A_1, A_2, A_3, A_4 表示年龄、有工作、有自己的房子合信贷情况 4 个特征，并以 1，2，3 表示年龄的值为青年、中年和老年，1，2 表示有工作和有自己的房子，以 1，2，3 表示信贷情况的值为非常好，好和一般。

求特征 A_1 的基尼系数:

$$Gini(D, A_1 = 1) = \frac{5}{15}(2 \times \frac{2}{5} \times (1 - \frac{2}{5})) + \frac{10}{15}(2 \times \frac{7}{10} \times (1 - \frac{7}{10})) = 0.44$$

$$Gini(D, A_1 = 2) = 0.48$$

$$Gini(D, A_1 = 3) = 0.44$$

由于 $Gini(D, A_1 = 1)$ 和 $Gini(D, A_1 = 3)$ 相等，且最小，所以 $A_1 = 1$ 和 $A_1 = 3$ 都可以作为选作 A_1 的最优切分点。

求特征 A_2 和 A_3 的基尼系数:

$$Gini(D, A_2 = 1) = 0.32$$

$$Gini(D, A_3 = 1) = 0.27$$

由于 A_2 和 A_3 只有一个切分点，所以它们就是最优切分点。

求特征 A_4 的基尼系数:

$$Gini(D, A_4 = 1) = 0.36$$

$$Gini(D, A_4 = 2) = 0.47$$

$$Gini(D, A_4 = 3) = 0.32$$

$Gini(D, A_4 = 3)$ 最小，所以 $A_4 = 3$ 为 A_4 的最优切分点。

在 A_1, A_2, A_3, A_4 几个特征中，$Gini(D, A_3 = 1)$ 最小，所以特征值 A_3 为最优特征，$A_3 = 1$ 为其最优划分点。于是根节点生成两个子结点，一个是叶结点。对另一个结点继续使用以上方法在 A_1, A_2, A_4 中选择最优特征及其最优切分点，结果是 $A_2 = 1$。依照此计算得知，所得结点都是叶子结点。

对于本问题，按照 CART 算法所得的决策树和按照 ID3 算法所生成的决策树完全一致。

3.1　python 代码实现

首先计算基尼系数的值:

```
#计算基尼系数
def gini(data):
    data_label=data.iloc[:,-1]
    label_num = data_label.value_counts()  # 有几类，每一类的数量
    res=0
    for k in label_num.keys():
```

```
        p_k=label_num[k]/len(data_label)
        res+=p_k**2
    gini_value=1-res
    return gini_value
gini_value=gini(data)
print('数据集的纯度—基尼值G(D)为:\n',gini_value)
print('=='*30)
```

我们数据集 D 的纯度的基尼值为:

```
============================================================
数据集的纯度—基尼值G(D)为:
 0.4982698961937716
============================================================
```

基尼值 $Gini(D)$：从数据集 D 中随机抽取两个样本，其类别标记不一致的概率。故，$Gini(D)$ 值越小，数据集 D 的纯度越高。

接下来我们来计算每个特征取值的基尼指数，以便找出最优切分点。

```
def gini_index(data,a):
    feature_class=data[a].value_counts()
    res=[]
    for v in feature_class.keys():
        weight=feature_class[v]/data.shape[0]
        gini_value=gini(data.loc[data[a]==v])
        #res+=weight*gini_value
        res.append([v,weight*gini_value])
    res=sorted(res,key=lambda x:x[-1])
    return res
for a in ['色泽','根蒂','敲声','纹理','脐部','触感']:
    res=gini_index(data,a)
    print('%s的基尼指数为:\n'%a, res)
    print('=='*30)
```

我们得到的结果为:

```
============================================================
色泽的基尼指数为:
 [['浅白', 0.09411764705882349], ['乌黑', 0.1568627450980392], ['青绿', 0.
                                        17647058823529413]]
============================================================
```

```
根蒂的基尼指数为:
 [['硬挺', 0.0], ['稍蜷', 0.2016806722689076], ['蜷缩', 0.22058823529411764
                                        ]]
================================================================
敲声的基尼指数为:
 [['清脆', 0.0], ['沉闷', 0.1411764705882353], ['浊响', 0.2823529411764706]
                                        ]
================================================================
纹理的基尼指数为:
 [['模糊', 0.0], ['稍糊', 0.09411764705882349], ['清晰', 0.1830065359477124
                                        ]]
================================================================
脐部的基尼指数为:
 [['平坦', 0.0], ['凹陷', 0.1680672268907563], ['稍凹', 0.17647058823529413
                                        ]]
================================================================
触感的基尼指数为:
 [['软粘', 0.1411764705882353], ['硬滑', 0.35294117647058826]]
================================================================
```

获取标签最多的那一类:

```
def get_most_label(data):
    data_label = data.iloc[:,-1]
    label_sort = data_label.value_counts(sort=True)
    return label_sort.keys()[0]
```

```
def get_most_label(label_list):
    return label_list.value_counts().idxmax()
```

挑选最优特征，即基尼指数最小的特征:

```
def get_best_feature(data):
    features = data.columns[:-1]
    res = {}
    for a in features:
        temp = gini_index(data, a)  #temp是列表, [feature_value, gini]
        res[a] = temp
    res = sorted(res.items(),key=lambda x:x[1][1])
    return res[0][0], res[0][1][0]
```

```
获取最佳划分特征:
 ('根蒂', '硬挺')
```

我们这里的 *res* 为:

```
[('色泽', [['浅白', 0.09411764705882349], ['乌黑', 0.1568627450980392], ['
                                        青绿', 0.17647058823529413]]), ('脐部
                                        ', [['平坦', 0.0], ['凹陷', 0.
                                        1680672268907563], ['稍凹', 0.
                                        17647058823529413]]), ('敲声', [['清
                                        脆', 0.0], ['沉闷', 0.
                                        1411764705882353], ['浊响', 0.
                                        2823529411764706]]), ('触感', [['软粘
                                        ', 0.1411764705882353], ['硬滑', 0.
                                        35294117647058826]]), ('纹理', [['模
                                        糊', 0.0], ['稍糊', 0.
                                        09411764705882349], ['清晰', 0.
                                        1830065359477124]]), ('根蒂', [['硬挺
                                        ', 0.0], ['稍蜷', 0.2016806722689076]
                                        , ['蜷缩', 0.22058823529411764]])]
```

```
res[0][0]=
 色泽
res[0][1][0]=
 ['浅白', 0.09411764705882349]
```

这里，我们总结一下 sorted 函数与 lambda 表达式结合的用法:

定理. *sorted()* 函数是全局排序函数，对所有可迭代的对象进行排序操作，它不会修改原对象，而将排序后的结果作为函数的返回值。

sort 与 *sorted* 的区别:

sort 是应用在 *list* 上的方法，*sorted* 可以对所有可迭代的对象进行排序操作。*list* 的 *sort* 方法返回的是对已经存在的列表进行操作，而内建函数 *sorted* 方法返回的是一个新的 *list*，而不是在原来的基础上进行的操作。

sorted 语法:

sorted(iterable, key=None, reverse=False)

参数说明:

iterable -可迭代对象。

key -主要是用来进行比较的元素，只有一个参数，具体的函数的参数就是取自于可迭代对象中，指定可迭代对象中的一个元素来进行排序。

reverse -排序规则，*reverse* = *True* 降序，*reverse* = *False* 升序（默认）。返回值：返回排序后的对象

lambda 表达式是 *Python* 中一类特殊的定义函数的形式，使用它可以定义一个匿名函数。与其它语言不同，*Python* 的 *lambda* 表达式的函数体只能有单独的一条语句，也就是返回值表达式语句。

1. 使用 *lambda* 表达式对一维数组进行倒序排序

```
nums = [2, 3, 0, 1, 5, 4]
nums1 = sorted(nums, key=lambda x:x*-1)
print(nums1)
```

```
[5, 4, 3, 2, 1, 0]
```

2. 按照二维矩阵下标为 *1* 的列进行排序

```
matrix = [[7,0],[4,4],[7,1],[5,0],[6,1],[5,2]]
ret = sorted(matrix, key=lambda x: x[1])
print(ret)
```

```
[[7, 0], [5, 0], [7, 1], [6, 1], [5, 2], [4, 4]]
```

3. 对字典数组的某一关键字进行排序

```
# 以 age 的降序进行排序
array = [{"age": 20, "name": "a"}, {"age": 25, "name": "b"}, {"age": 10, "
                                    name": "c"}]
array = sorted(array, key=lambda x:x["age"], reverse=True)
# 上一句等同于
# array = sorted(array, key=lambda x:-x["age"])
print(array)
```

```
[{'age': 25, 'name': 'b'}, {'age': 20, 'name': 'a'}, {'age':
                                    10, 'name': 'c'}]
```

4. 先按照成绩降序排序，相同成绩的按照名字升序排序

```
d1 = [{'name': 'alice', 'score': 38}, {'name': 'bob', 'score': 18}, {'name'
                                    : 'darl', 'score': 28}, {'name': '
                                    christ', 'score': 28}]
l = sorted(d1, key=lambda x:(-x['score'], x['name']))
```

```
print(l)
```

```
[{'name': 'alice', 'score': 38}, {'name': 'christ', 'score': 28
                                 }, {'name': 'darl', 'score': 28
                                 }, {'name': 'bob', 'score': 18}
                                 ]
```

我们定义的决策树为:

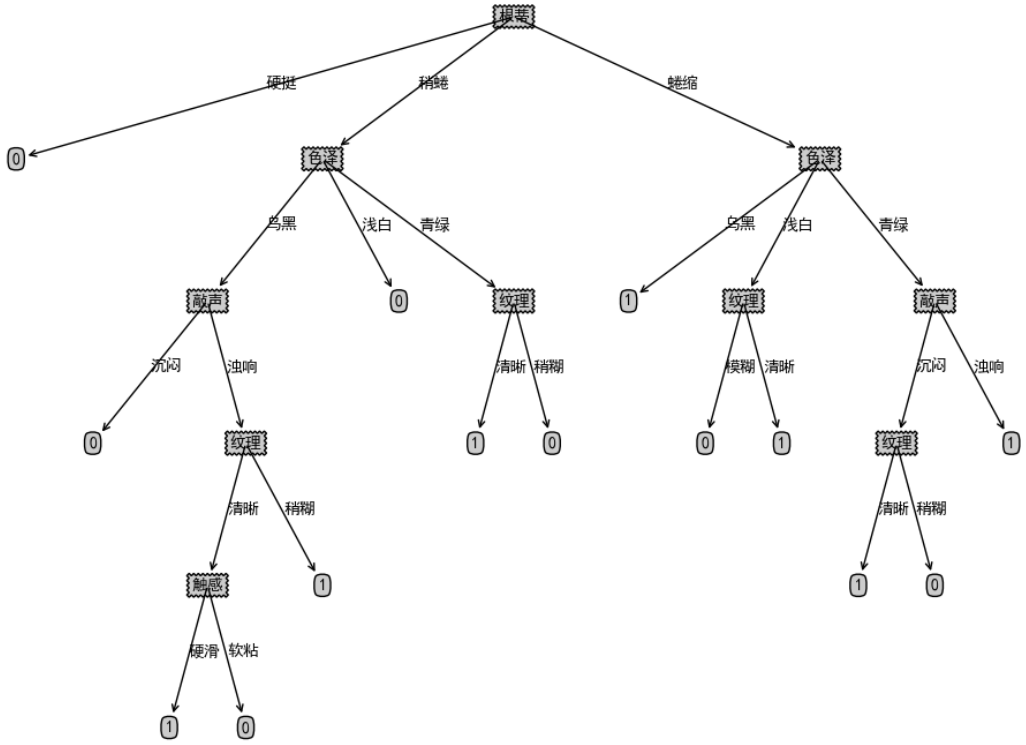
```
def create_tree(data):
    feature = data.columns[:-1]
    label_list = data.iloc[:, -1]
    #如果样本全属于同一类别C, 将此节点标记为C类叶节点
    if len(pd.unique(label_list)) == 1:
        return label_list.values[0]
    #如果待划分的属性集A为空, 或者样本在属性A上取值相同, 则把该节点作为叶节
                                        点, 并标记为样本数最多的分类
    elif len(feature)==0 or len(data.loc[:,feature].drop_duplicates())==1:
        return get_most_label(label_list)
    #从A中选择最优划分属性
    best_attr,lable = get_best_feature(data)
    tree = {best_attr: {}}
    #对于最优划分属性的每个属性值, 生成一个分支
    for attr,gb_data in data.groupby(by=best_attr):
        if len(gb_data) == 0:
            tree[best_attr][attr] = get_most_label(label_list)
        else:
            #在data中去掉已划分的属性
            new_data = gb_data.drop(best_attr,axis=1)
            #递归构造决策树
            tree[best_attr][attr] = create_tree(new_data)
    return tree
my_tree=create_tree(data)
print(my_tree)
```

输出的结果为:

```
{'根蒂': {'硬挺': 0, '稍蜷': {'色泽': {'乌黑': {'敲声': {'沉闷': 0, '浊响':
                                {'纹理': {'清晰': {'触感': {'硬滑':
                               1, '软粘': 0}}, '稍糊': 1}}}}, '浅白'
                               : 0, '青绿': {'纹理': {'清晰': 1, '稍
```

```
糊': 0}}}}, '蜷缩': {'色泽': {'乌黑':
 1, '浅白': {'纹理': {'模糊': 0, '清
晰': 1}}, '青绿': {'敲声': {'沉闷': {
'纹理': {'清晰': 1, '稍糊': 0}}, '浊
响': 1}}}}}}
```

可视化的代码与上述情况相同，这里不再赘述，直接套用, 生成的结果如下:



我们利用生成的决策树进行预测:

```
def predict(Tree , test_data):
    first_feature = list(Tree.keys())[0] #第一个特征
    second_dict = Tree[first_feature] #第一个特征后面的字典
    input_first = test_data.get(first_feature) #预测输入的第一个特征值是多
                                              少
    input_value = second_dict[input_first] #预测输入对应的字典
    if isinstance(input_value , dict): #判断分支还是不是字典
        class_label = predict(input_value, test_data)
    else:
        class_label = input_value
```

```
    return class_label

test_data = {'色泽':'青绿','根蒂':'蜷缩','敲声':'浊响','纹理':'稍糊','脐部'
                                        :'凹陷','触感':'硬滑'}
print('预测结果为:\n',predict(my_tree,test_data))
```

```
预测结果为:
 1
```

第四章　源代码

4.1　ID3 C4.5

```
import numpy as np
import pandas as pd
from math import log
import operator
import matplotlib.pyplot as plt
import matplotlib as mpl
path='./2022-06-19data/data_word.csv'
data=pd.read_csv(path)
print('数据集展示为:\n',data)
print(data.keys())

def cal_information_entropy(data):
    data_labels=data.iloc[:,-1]
    label_class=data_labels.value_counts() #标签计数
    entropy=0
    for key in label_class.keys():
        prob=label_class[key]/len(data_labels)
        entropy+=-prob*log(prob,2)
    return entropy
entropy=cal_information_entropy(data)
print('=='*30)
print('entropy=\n',entropy)
print('=='*30)

#计算给定数据属性a的信息增益
def cal_information_gain(data,a):
    entropy = cal_information_entropy(data)
    feature_class = data[a].value_counts()#特征有多少种可能
```

```
    #print(feature_class)
    #print('特征类型:\n', feature_class.keys())
    #print('==' * 30)
    gain_entropy = 0.0
    for v in feature_class.keys():
        weight = feature_class[v]/data.shape[0]
        entropy_v= cal_information_entropy(data.loc[data[a] == v])
        condition_entropy=weight*entropy_v #条件熵的每一项
        gain_entropy+=weight*entropy_v
        #print('权重值为:\n',weight)
        #print('对应的熵值为:\n',entropy_v)
        #print('对应的条件熵的子集为:\n',condition_entropy)
        #print(data.loc[data[a] == v],type(data.loc[data[a] == v]))
        #print('=='*30)
    infomation_entropy_gain = entropy - gain_entropy
    return infomation_entropy_gain
"""
print('色泽的信息增益为:\n',cal_information_gain(data,'色泽'))
print('==' * 30)
print('根蒂的信息增益为:\n',cal_information_gain(data,'根蒂'))
print('==' * 30)
print('敲声的信息增益为:\n',cal_information_gain(data,'敲声'))
print('==' * 30)
print('纹理的信息增益为:\n',cal_information_gain(data,'纹理'))
print('==' * 30)
print('脐部的信息增益为:\n',cal_information_gain(data,'脐部'))
print('==' * 30)
print('触感的信息增益为:\n',cal_information_gain(data,'触感'))
print('==' * 30)
"""

def cal_gain_ratio(data , a):
    #先计算固有值intrinsic_value
    IV_a = 0
    feature_class = data[a].value_counts()  # 特征有多少种可能
    for v in feature_class.keys():
        weight = feature_class[v]/data.shape[0]
        IV_a += -weight*log(weight,2)
    gain_ration = cal_information_gain(data,a)/IV_a
```

```
    return gain_ration

"""
#ID3算法，挑选信息增益最大的特征
def get_best_feature(data):
    features = data.columns[:-1]
    res = {}
    for a in features:
        info_gain = cal_information_gain(data,a)
        res[a] = info_gain
    res = sorted(res.items(),key=lambda x:x[1],reverse=True)
    return res[0][0]
print('获取分类的最佳特征:\n',get_best_feature(data))

"""
def get_best_feature(data):
    features = data.columns[:-1]
    res = {}
    for a in features:
        temp = cal_information_gain(data, a)
        gain_ration = cal_gain_ratio(data,a)
        res[a] = (temp,gain_ration)
    res = sorted(res.items(),key=lambda x:x[1][0],reverse=True) #按信息增益
                                            排名
    res_avg = sum([x[1][0] for x in res])/len(res) #信息增益平均水平
    good_res = [x for x in res if x[1][0] >= res_avg] #选取信息增益高于平均
                                            水平的特征
    result =sorted(good_res,key=lambda x:x[1][1],reverse=True) #将信息增益
                                            高的特征按照增益率进行排名
    return result[0][0] #返回高信息增益中增益率最大的特征
print('获取最佳的划分属性:\n',get_best_feature(data))

#获取标签最多的那一类
def get_most_label(data):
    data_label = data.iloc[:,-1]
    label_sort = data_label.value_counts(sort=True)
    return label_sort.keys()[0]
print('获取标签最多的一类为:\n',get_most_label(data))
```

```
#将数据转化为（属性值：数据）的元组形式返回，并删除之前的特征列
def drop_exist_feature(data, best_feature):
    attribute = pd.unique(data[best_feature])
    new_data = [(nd, data[data[best_feature] == nd]) for nd in attribute]
    new_data = [(n[0], n[1].drop([best_feature], axis=1)) for n in new_data
                                                ]
    return new_data

column_count = dict([(ds, list(pd.unique(data[ds]))) for ds in data.iloc[:,
                                            :-1].columns])
print(column_count)

"""
函数说明:创建决策树
Parameters:
    dataSet - 训练数据集
    labels - 分类属性标签
    featLabels - 存储选择的最优特征标签
Returns:
    myTree - 决策树
"""
def create_tree(data):
    data_label = data.iloc[:,-1] #取分类标签
    if len(data_label.value_counts()) == 1: #只有一类
        return data_label.values[0]
    if all(len(data[i].value_counts()) == 1 for i in data.iloc[:,:-1].
                                            columns): #所有数据的特征值一样，
                                            选样本最多的类作为分类结果
        return get_most_label(data)
    best_feature = get_best_feature(data) #根据信息增益得到的最优划分特征
    Tree = {best_feature:{}} #用字典形式存储决策树
    exist_vals = pd.unique(data[best_feature]) #当前数据下最佳特征的取值
    if len(exist_vals) != len(column_count[best_feature]): #如果特征的取值
                                            相比于原来的少了
        no_exist_attr = set(column_count[best_feature]) - set(exist_vals) #
                                                少的那些特征
        for no_feat in no_exist_attr:
            Tree[best_feature][no_feat] = get_most_label(data) #缺失的特征
                                                    分类为当前类别最多的
```

```
    for item in drop_exist_feature(data,best_feature): #根据特征值的不同递
                                                归创建决策树
        Tree[best_feature][item[0]] = create_tree(item[1])
    return Tree

my_tree=create_tree(data)
print('构造的决策树如下:\n',my_tree)

def plot_node(node_txt, center_pt, parent_pt, node_type):
    arrow_args = dict(arrowstyle="<-") #定义箭头格式
    # 绘制结点
    create_plot.ax1.annotate(node_txt, xy=parent_pt, xycoords='axes
                                            fraction',xytext=center_pt,
                                            textcoords='axes fraction',va="
                                            center", ha="center", bbox=
                                            node_type, arrowprops=arrow_args)
"""
函数说明:获取决策树叶子结点的数目
Parameters:
    myTree - 决策树
Returns:
    numLeafs - 决策树的叶子结点的数目
"""
def get_num_leafs(my_tree):
    num_leafs = 0 #初始化叶子
    first_str = list(my_tree.keys())[0] ##python3中my——tree.keys()返回的
                                            是dict_keys,不在是list,所以不能使
                                            用myTree.keys()[0]的方法获取结点
                                            属性，可以使用list(myTree.keys())
                                            [0]
    second_dict = my_tree[first_str] ##获取下一组字典
    for key in second_dict.keys():
        if type(second_dict[key]).__name__ == 'dict': #测试该结点是否为字
                                                典，如果不是字典，代表此结点
                                                为叶子结点
            num_leafs += get_num_leafs(second_dict[key])
        else:
            num_leafs += 1
```

```
    return num_leafs

"""
函数说明:获取决策树的层数
Parameters:
    myTree - 决策树
Returns:
    maxDepth - 决策树的层数
"""
def get_tree_depth(my_tree):
    max_depth = 0 #初始化决策树深度
    first_str = list(my_tree.keys())[0]
    second_dict = my_tree[first_str] #获取下一个字典
    for key in second_dict.keys():
        if type(second_dict[key]).__name__ == 'dict': #测试该结点是否为字
                                                      典，如果不是字典，代表此结点
                                                      为叶子结点
            thisDepth = get_tree_depth(second_dict[key]) + 1
        else:
            thisDepth = 1
        if thisDepth > max_depth: #更新层数
            max_depth = thisDepth
    return max_depth

"""
函数说明:标注有向边属性值
Parameters:
    cntr_pt、parent_pt - 用于计算标注位置
    txt_string - 标注的内容
Returns:
    无
"""
def plot_mid_text(cntr_pt, parent_pt, txt_string):
    x_mid = (parent_pt[0] - cntr_pt[0]) / 2.0 + cntr_pt[0]
    y_mid = (parent_pt[1] - cntr_pt[1]) / 2.0 + cntr_pt[1]
    create_plot.ax1.text(x_mid, y_mid, txt_string)

"""
函数说明:绘制决策树
```

```
    Parameters:
        my_tree - 决策树(字典)
        parent_pt - 标注的内容
        node_txt - 结点名
    Returns:
        无
    """

    leaf_node = dict(boxstyle="round4", fc="0.8")
    decision_node = dict(boxstyle="sawtooth", fc="0.8")

    def plot_tree(my_tree, parent_pt, node_txt):
        num_leafs = get_num_leafs(my_tree)
        depth = get_tree_depth(my_tree)
        first_str = list(my_tree.keys())[0]
        cntr_pt = (plot_tree.x_off + (1.0 + float(num_leafs)) / 2.0 / plot_tree
                                              .total_w, plot_tree.y_off)
        plot_mid_text(cntr_pt, parent_pt, node_txt)
        plot_node(first_str, cntr_pt, parent_pt, decision_node)
        second_dict = my_tree[first_str]
        plot_tree.y_off = plot_tree.y_off - 1.0 / plot_tree.total_d
        for key in second_dict.keys():
            if type(second_dict[key]).__name__ == 'dict':
                plot_tree(second_dict[key], cntr_pt, str(key))
            else:
                plot_tree.x_off = plot_tree.x_off + 1.0 / plot_tree.total_w
                plot_node(second_dict[key], (plot_tree.x_off, plot_tree.y_off),
                                                       cntr_pt, leaf_node)
                plot_mid_text((plot_tree.x_off, plot_tree.y_off), cntr_pt, str(
                                                      key))
        plot_tree.y_off = plot_tree.y_off + 1.0 / plot_tree.total_d

    """
    函数说明:创建绘制面板
    Parameters:
        in_tree - 决策树(字典)
    Returns:
        无
    """
```

```
mpl.rcParams['font.family'] = 'SimHei'
plt.rcParams['axes.unicode_minus'] = False
def create_plot(in_tree):
    fig = plt.figure(1,figsize=(12,8),facecolor='white')
    fig.clf()
    axprops = dict(xticks=[], yticks=[])
    create_plot.ax1 = plt.subplot(111, frameon=False, **axprops)
    plot_tree.total_w = float(get_num_leafs(in_tree))
    plot_tree.total_d = float(get_tree_depth(in_tree))
    plot_tree.x_off = -0.5 / plot_tree.total_w
    plot_tree.y_off = 1.0
    plot_tree(in_tree, (0.5, 1.0), '')
    plt.show()
print(create_plot(my_tree))

def predict(Tree , test_data):
    first_feature = list(Tree.keys())[0]
    second_dict = Tree[first_feature]
    input_first = test_data.get(first_feature)
    input_value = second_dict[input_first]
    if isinstance(input_value , dict): #判断分支还是不是字典
        class_label = predict(input_value, test_data)
    else:
        class_label = input_value
    return class_label

def predict(Tree , test_data):
    first_feature = list(Tree.keys())[0]
    second_dict = Tree[first_feature]
    input_first = test_data.get(first_feature)
    input_value = second_dict[input_first]
    if isinstance(input_value , dict): #判断分支还是不是字典
        class_label = predict(input_value, test_data)
    else:
        class_label = input_value
    return class_label
test_data = {'色泽':'青绿','根蒂':'蜷缩','敲声':'浊响','纹理':'稍糊','脐部'
                                        :'凹陷','触感':'硬滑'}
print('预测结果为:\n',predict(my_tree,test_data))
```

4.2 CART

```
import pandas as pd
import numpy as np
from math import log
import operator
import matplotlib.pyplot as plt
import matplotlib as mpl
path='./2022-06-19data/data_word.csv'
data=pd.read_csv(path)
print('数据集展示为:\n',data)
print(data.keys())
print('=='*30)

#计算基尼系数
def gini(data):
    data_label=data.iloc[:,-1]
    label_num = data_label.value_counts()  # 有几类，每一类的数量
    res=0
    for k in label_num.keys():
        p_k=label_num[k]/len(data_label)
        res+=p_k**2
    gini_value=1-res
    return gini_value
gini_value=gini(data)
print('数据集的纯度—基尼值G(D)为:\n',gini_value)
print('=='*30)

# 计算每个特征取值的基尼指数，找出最优切分点
def gini_index(data,a):
    feature_class=data[a].value_counts()
    res=[]
    for v in feature_class.keys():
        weight=feature_class[v]/data.shape[0]
        gini_value=gini(data.loc[data[a]==v])
        #res+=weight*gini_value
        res.append([v,weight*gini_value])
    res=sorted(res,key=lambda x:x[-1])
    return res[0]
```

```
for a in ['色泽','根蒂','敲声','纹理','脐部','触感']:
    res=gini_index(data,a)
    print('%s的基尼指数为:\n'%a, res)
    print('=='*30)

#获取标签最多的那一类
def get_most_label(data):
    data_label = data.iloc[:,-1]
    label_sort = data_label.value_counts(sort=True)
    return label_sort.keys()[0]

#挑选最优特征，即基尼指数最小的特征
def get_best_feature(data):
    features = data.columns[:-1]
    res = {}
    for a in features:
        temp = gini_index(data, a)  #temp是列表，【feature_value, gini】
        res[a] = temp
    res = sorted(res.items(),key=lambda x:x[1][1]) #按照res[1][1]进行排序
    return res[0][0], res[0][1][0]
print('获取最佳划分特征:\n',get_best_feature(data))
print('=='*30)

def drop_exist_feature(data, best_feature, value, type):
    attr = pd.unique(data[best_feature]) #表示特征所有取值的数组
    if type == 1: #使用特征==value的值进行划分
        new_data = [[value], data.loc[data[best_feature] == value]]
    else:
        new_data = [attr, data.loc[data[best_feature] != value]]
    new_data[1] = new_data[1].drop([best_feature], axis=1) #删除该特征
    return new_data

#创建决策树
def get_most_label(label_list):
    return label_list.value_counts().idxmax()

# 创建决策树，传入的是一个dataframe，最后一列为label
def create_tree(data):
```

```
    feature = data.columns[:-1]
    label_list = data.iloc[:, -1]
    #如果样本全属于同一类别C，将此节点标记为C类叶节点
    if len(pd.unique(label_list)) == 1:
        return label_list.values[0]
    #如果待划分的属性集A为空，或者样本在属性A上取值相同，则把该节点作为叶节
                                        点，并标记为样本数最多的分类
    elif len(feature)==0 or len(data.loc[:,feature].drop_duplicates())==1:
        return get_most_label(label_list)
    #从A中选择最优划分属性
    best_attr,lable = get_best_feature(data)
    tree = {best_attr: {}}
    #对于最优划分属性的每个属性值，生成一个分支
    for attr,gb_data in data.groupby(by=best_attr):
        if len(gb_data) == 0:
            tree[best_attr][attr] = get_most_label(label_list)
        else:
            #在data中去掉已划分的属性
            new_data = gb_data.drop(best_attr,axis=1)
            #递归构造决策树
            tree[best_attr][attr] = create_tree(new_data)
    return tree
my_tree=create_tree(data)
print(my_tree)

def plot_node(node_txt, center_pt, parent_pt, node_type):
    arrow_args = dict(arrowstyle="<-") #定义箭头格式
    # 绘制结点
    create_plot.ax1.annotate(node_txt, xy=parent_pt, xycoords='axes
                                        fraction',xytext=center_pt,
                                        textcoords='axes fraction',va="
                                        center", ha="center", bbox=
                                        node_type, arrowprops=arrow_args)
"""
函数说明:获取决策树叶子结点的数目
Parameters:
    myTree - 决策树
Returns:
    numLeafs - 决策树的叶子结点的数目
```

```
"""
def get_num_leafs(my_tree):
    num_leafs = 0 #初始化叶子
    first_str = list(my_tree.keys())[0] ##python3中my——tree.keys()返回的
                                        是dict_keys,不在是list,所以不能使
                                        用myTree.keys()[0]的方法获取结点
                                        属性，可以使用list(myTree.keys())
                                        [0]
    second_dict = my_tree[first_str] ##获取下一组字典
    for key in second_dict.keys():
        if type(second_dict[key]).__name__ == 'dict': #测试该结点是否为字
                                                      典，如果不是字典，代表此结点
                                                      为叶子结点
            num_leafs += get_num_leafs(second_dict[key])
        else:
            num_leafs += 1
    return num_leafs

"""
函数说明:获取决策树的层数
Parameters:
    myTree - 决策树
Returns:
    maxDepth - 决策树的层数
"""
def get_tree_depth(my_tree):
    max_depth = 0 #初始化决策树深度
    first_str = list(my_tree.keys())[0]
    second_dict = my_tree[first_str] #获取下一个字典
    for key in second_dict.keys():
        if type(second_dict[key]).__name__ == 'dict': #测试该结点是否为字
                                                      典，如果不是字典，代表此结点
                                                      为叶子结点
            thisDepth = get_tree_depth(second_dict[key]) + 1
        else:
            thisDepth = 1
        if thisDepth > max_depth: #更新层数
            max_depth = thisDepth
    return max_depth
```

```
    """
    函数说明:标注有向边属性值
    Parameters:
        cntr_pt、parent_pt - 用于计算标注位置
        txt_string - 标注的内容
    Returns:
        无
    """
    def plot_mid_text(cntr_pt, parent_pt, txt_string):
        x_mid = (parent_pt[0] - cntr_pt[0]) / 2.0 + cntr_pt[0]
        y_mid = (parent_pt[1] - cntr_pt[1]) / 2.0 + cntr_pt[1]
        create_plot.ax1.text(x_mid, y_mid, txt_string)

    """
    函数说明:绘制决策树
    Parameters:
        my_tree - 决策树(字典)
        parent_pt - 标注的内容
        node_txt - 结点名
    Returns:
        无
    """

    leaf_node = dict(boxstyle="round4", fc="0.8")
    decision_node = dict(boxstyle="sawtooth", fc="0.8")

    def plot_tree(my_tree, parent_pt, node_txt):
        num_leafs = get_num_leafs(my_tree)
        depth = get_tree_depth(my_tree)
        first_str = list(my_tree.keys())[0]
        cntr_pt = (plot_tree.x_off + (1.0 + float(num_leafs)) / 2.0 / plot_tree
                                                .total_w, plot_tree.y_off)
        plot_mid_text(cntr_pt, parent_pt, node_txt)
        plot_node(first_str, cntr_pt, parent_pt, decision_node)
        second_dict = my_tree[first_str]
        plot_tree.y_off = plot_tree.y_off - 1.0 / plot_tree.total_d
        for key in second_dict.keys():
            if type(second_dict[key]).__name__ == 'dict':
```

```
            plot_tree(second_dict[key], cntr_pt, str(key))
        else:
            plot_tree.x_off = plot_tree.x_off + 1.0 / plot_tree.total_w
            plot_node(second_dict[key], (plot_tree.x_off, plot_tree.y_off),
                                                    cntr_pt, leaf_node)
            plot_mid_text((plot_tree.x_off, plot_tree.y_off), cntr_pt, str(
                                                   key))
    plot_tree.y_off = plot_tree.y_off + 1.0 / plot_tree.total_d

"""
函数说明:创建绘制面板
Parameters:
    in_tree - 决策树(字典)
Returns:
    无
"""

mpl.rcParams['font.family'] = 'SimHei'
plt.rcParams['axes.unicode_minus'] = False

def create_plot(in_tree):
    fig = plt.figure(1,figsize=(12,8),facecolor='white')
    fig.clf()
    axprops = dict(xticks=[], yticks=[])
    create_plot.ax1 = plt.subplot(111, frameon=False, **axprops)
    plot_tree.total_w = float(get_num_leafs(in_tree))
    plot_tree.total_d = float(get_tree_depth(in_tree))
    plot_tree.x_off = -0.5 / plot_tree.total_w
    plot_tree.y_off = 1.0
    plot_tree(in_tree, (0.5, 1.0), '')
    plt.show()
print(create_plot(my_tree))

def predict(Tree , test_data):
    first_feature = list(Tree.keys())[0]
    second_dict = Tree[first_feature]
    input_first = test_data.get(first_feature)
    input_value = second_dict[input_first]
    if isinstance(input_value , dict): #判断分支还是不是字典
```

```
        class_label = predict(input_value, test_data)
    else:
        class_label = input_value
    return class_label


test_data = {'色泽':'青绿','根蒂':'蜷缩','敲声':'浊响','纹理':'稍糊','脐部'
                                        :'凹陷','触感':'硬滑'}
print('预测结果为:\n',predict(my_tree,test_data))
```