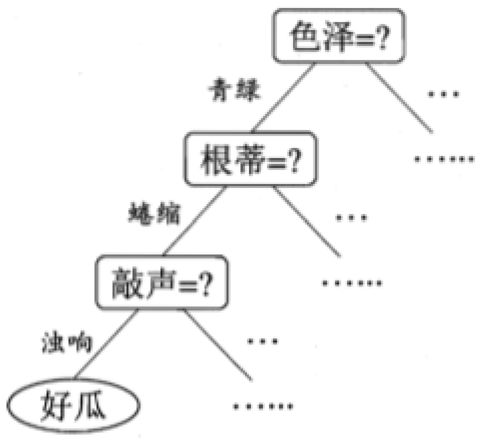
决策树

一、 直观理解

此例子来自于周志华《机器学习》。



我们在买西瓜的时候，要判断一个西瓜是不是好西瓜，首先看色泽，如果色泽=青绿，接下来看根蒂。如果根蒂=蜷缩，再听敲击声。如果悄声=浊响，则判断其为好瓜。

我们使判断流程化以后，就有了左图的决策树，以后判断一个西瓜是否为好瓜，只需要输入（色泽，根蒂，敲声）三个特征，即可以得出结果。

接下来我们讨论两个问题：

1. 最优划分属性。色泽、根蒂、敲声三个特征，哪一个特征对我们判断一个西瓜是不是好瓜帮助最大？我们假设：

① “色泽=青绿”的情形下，有 99 个样本是好瓜，有 1 个样本是坏瓜。

② “敲声=浊响”的情形下，有 50 个样本是好瓜，有 50 个样本是坏瓜。

如果我们只知道“色泽=青绿”，有很大概率判断正确。但是只知道“敲声=浊响”的话，我们却几乎无法判断一个西瓜是好瓜还是坏瓜。因此，我们认为“色泽=青绿”这个分支节点对我们进行判断更加重要，纯度（purity）越高。

那么，纯度可以度量吗？答案是可以，常用信息熵（information entropy）来度量信息纯度。

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

其中，p_k为集合 D 中第 k 类样本所占的比例。

在①例中，$\operatorname{Ent}\left(\text{色泽}=\text{青绿}\right)=-0.99\log_2 0.99-0.01\log_2 0.01=0.08$

在②例中，$\operatorname{Ent}\left(\text{敲声}=\text{浊响}\right)=-0.50\log_2 0.50-0.50\log_2 0.50=1.00$

Ent(D)越小，纯度越高。也就是说，不确定性越小，纯度越高。

整个“色泽”特征的纯度是多少？ “色泽=青绿”的纯度我们已经计算出来，接下来计算“色泽=乌黑”、“色泽=浅白”下的纯度。假设 Ent(色泽=青绿) =0.08，样例为 100 个，Ent(色泽=乌黑) =0.05，样例为 100 个，Ent(色泽=浅白) =0.72，样例为 200 个。则总纯度为 Ent（总样本，色泽）= 100/400*0.08+100/400*0.05+200/400*0.72=0.3925， 如果计算出 Ent (总样本, 敲声) =0.45，则我们认为色泽更加重要，应该成为第一分支。

2. 信息增益（information gain）

假设 400 个总样本中，有 200 个好瓜，200 个坏瓜。则$\operatorname{Ent}\left(\text{总样本}\right)=1.00$

色泽属性的信息增益为 Gain（总样本，色泽）=1.00-0.3925=0.6075

敲声属性的信息增益为 Gain（总样本，敲声）=1.00-0.45=0.55

色泽的信息增益更大，于是我们认为色泽这个特征更加重要。

那么“色泽=青绿”下面，敲声和根蒂哪个特征更重要?

计算 Gain（色泽=青绿，敲声)、Gain（色泽=青绿，根蒂)，然后比较即可。

同理，“色泽=乌黑”下面，计算 Gain（色泽=乌黑，敲声)、Gain（色泽=乌黑，根蒂)，然后比较。

……直到生成完全决策树。

常用的算法有 ID3 算法，第二节里 Python 代码实现的就是 ID3 算法。

3. 增益率（gain ratio）

假设有 17 个西瓜，每个西瓜都有一个特殊的编号，分别从 1-17。则 Ent(编号=1) = Ent(编号=2) =……= Ent(编号=17) =0。Gain（总样本，编号）=1.00-0=1.00，比色泽、敲声特征的信息增益要大，则 ID3 算法会将编号作为第一个分类特征。

这样会导致一个问题，在“编号”这个特征下，每个分支节点仅包含一个样本，这样的决策树没有泛化能力。因为我们无法凭借编号来评判一个瓜是好瓜还是坏瓜。

所以，在 C4.5 算法中，使用增益率来选择最优划分属性。

Gain_ratio（D,a） = $\dfrac{Gain(D,a)}{IV(a)}$　其中$IV(a)=-\sum_{v=1}^{V}\frac{|D^v|}{|D|}\log_2\frac{|D^v|}{|D|}$

IV（编号） = $-17\times\frac{1}{17}\log_2\frac{1}{17}$ = 4.08

Gain（总样本，编号）= 0.245

于是增益率可以减少对可取值数目较大的属性特征的偏好。

4. 基尼指数

CART 决策树（另一种决策树）使用 “基尼指数” （Gini index）选择划分属性。

$\mathrm{Gini}(\mathrm{D})=1-\sum_{k=1}^{|y|}p_k^2$ 直观上来理解，Gini(D)反映了从数据集 D 中随机抽取两个样本，其类别标记不一致的概率。数值越小，纯度越高，与信息熵$\mathrm{Ent}(D)$相似。比如总样本由 99 个好瓜和 1 个坏瓜时组成，

$\mathrm{Gini}\left(总样本\right)=1-(0.99^2+0.01^2)\ =0.0198$

50 个好瓜和 50 个坏瓜时，$\mathrm{Gini}\left(总样本\right)=1-(0.50^2+0.50^2)=0.50$。

Gini_index（D,a）$=\sum_{v=1}^{V}\frac{|D^v|}{|D|}\mathrm{Gini}(\mathrm{D}^v)$ 很简单的一个加权平均，与计算$\mathrm{Ent}(D,a)$差不多。

二、Python 实现

1. 信息熵的计算

```
def calcShannonEnt(dataSet):
    numEntries = len(dataSet)
    labelCounts = {}
    for featVec in dataSet:
        currentLabel = featVec[-1]
        if currentLabel not in 
labelCounts.keys():labelCounts[currentLabel] = 0
        labelCounts[currentLabel] += 1
    shannonEnt = 0.0
    for key in labelCounts:
        prob = float(labelCounts[key])/numEntries
        shannonEnt -= prob * log(prob,2)
    return shannonEnt
```

使用一个简单的集合测试

```python
def createDataSet():
    dataSet = [[1,1,'yes'],[1,1,'yes'],
               [1,0,'no'],[0,1,'no'],[0,1,'no']]
    labels = ['no surfacing','flippers']
    return dataSet,labels

myDat,labels = createDataSet()
print(calcShannonEnt(myDat))
```

该集合信息熵为 0.9709505944546686

我们修改一下测试的集合，发现此时的结果为 1.3709505944546687，因为不确定增大，所以信息熵增大。

```python
def createDataSet():
    dataSet = [[1,1,'maybe'],[1,1,'yes'],
               [1,0,'no'],[0,1,'no'],[0,1,'no']]
    labels = ['no surfacing','flippers']
    return dataSet,labels

myDat,labels = createDataSet()
print(calcShannonEnt(myDat))
```

2. 生成分支

```python
def splitDataSet(dataSet,axis,value):
    retDataSet = []
    for featVec in dataSet:
        if featVec[axis] == value:
            # 去掉该结点的值，并保留所有子结点的值
            reducedFeatVec = featVec[:axis]
            reducedFeatVec.extend(featVec[axis+1:])
            retDataSet.append(reducedFeatVec)
    return retDataSet
```

测试：将之前的集合以第一个特征按 0,1 分为两类

```
myDat,labels = createDataSet()
print(splitDataSet(myDat,0,1))
print(splitDataSet(myDat,0,0))
```

结果如下，即生成了一个简单的决策树

```
[[1, 'maybe'], [1, 'yes'], [0, 'no']]
[[1, 'no'], [1, 'no']]
```

3. 选择最优划分属性

```
def chooseBestFeatureToSplit(dataSet):
    numFeatures = len(dataSet[0]) - 1
    baseEntropy = calcShannonEnt(dataSet)
    bestInfoGain = 0.0; bestFeature = -1
    for i in range(numFeatures):
        featList = [example[i] for example in dataSet]
        uniqueVals = set(featList)
        newEntropy = 0.0
        for value in uniqueVals:
            subDataSet = splitDataSet(dataSet,i,value)
            prob = len(subDataSet)/float(len(dataSet))
            newEntropy += prob * calcShannonEnt(subDataSet)
        infoGain = baseEntropy - newEntropy
        if(infoGain > bestInfoGain):
            bestInfoGain = infoGain
            bestFeature = i
    return bestFeature
```

测试：

```
myDat,labels = createDataSet()
print(chooseBestFeatureToSplit(myDat))
```

结果为 0，即第一个特征为最佳的分离点。

4. 通过递归生成完全决策树

```python
# 只剩下一个特征的时候分裂结束，返回多数的一个结果作为该枝杈的末尾
def majorityCnt(classList):
    classCount = {}
    for vote in classList:
        if vote not in classCount.keys(): classCount[vote] = 0
        classCount[vote] += 1
    sortedClassCount = sorted(classCount.items(),
                              key=operator.itemgetter(1),reverse=True)
    return sortedClassCount[0],[0]

def createTree(dataSet,labels):
    classList = [example[-1] for example in dataSet]
    # 所有特征分类对应的结果相同时，分裂结束
    if classList.count(classList[0]) == len(classList):
        return classList[0]
    # 只剩下一个特征的时候分裂结束，返回多数的一个结果作为该枝杈的末尾
    if len(dataSet[0]) == 1:
        return majorityCnt(classList)
    bestFeat = chooseBestFeatureToSplit(dataSet)
    bestFeatLabel = labels[bestFeat]
    myTree = {bestFeatLabel:{}}
    del(labels[bestFeat])
    featValues = [example[bestFeat] for example in dataSet]
    uniqueVals = set(featValues)
    # 通过递归形成完整的决策树
    for value in uniqueVals:
        subLabels = labels[:]
        myTree[bestFeatLabel][value] = createTree(splitDataSet\
                (dataSet,bestFeat,value),subLabels)
    return myTree
```

测试

```python
myDat,labels = createDataSet()

print(createTree(myDat,labels))
```

结果

```
{'no surfacing': {0: 'no', 1: {'flippers': {0: 'no', 1: 'yes'}}}}
```

5. 使用二叉树进行分类

```
def classify(inputTree,featLabels,testVec):
    firstStr = list(inputTree.keys())[0]     # 确定匹配的特征
    secondDict = inputTree[firstStr]
    featIndex = featLabels.index(firstStr)
    key = testVec[featIndex]
    valueOfFeat = secondDict[key]    # 找到该特征下实例应归属的分支

    if isinstance(valueOfFeat, dict):    # 如果返回的是字典类型，递归查找至决策树末尾
        classLabel = classify(valueOfFeat, featLabels, testVec)
    else: classLabel = valueOfFeat
    return classLabel
```

测试

```
myDat,labels = createDataSet()
clabels = labels.copy()
data = createTree(myDat,labels)
print(classify(data,clabels,[1,0]))
```

结果

```
no
```

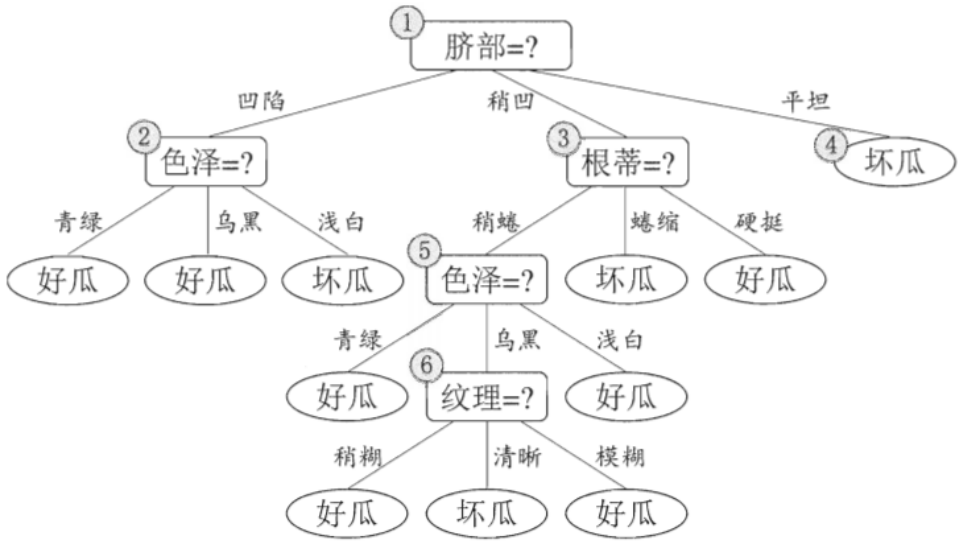
三、　算法深入

1. 剪枝处理

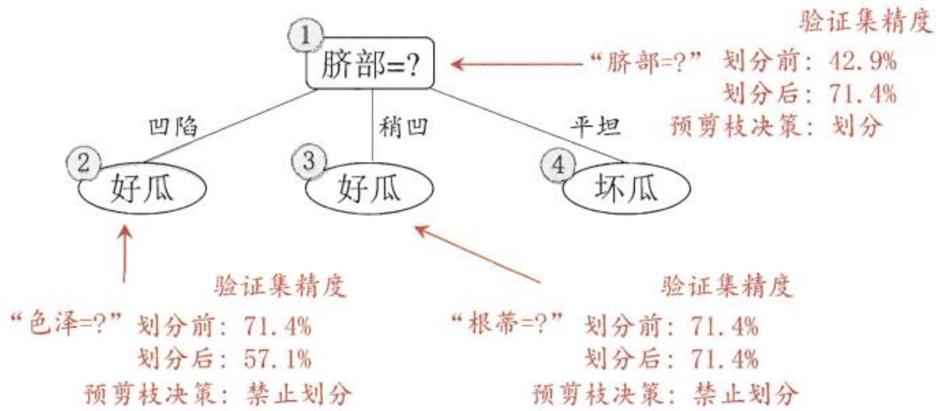
决策树会产生“过拟合”风险，即决策树十分复杂，缺乏泛化能力。于是我们通过减少决策树的分支数量来降低过拟合风险。决策树剪枝可分为预剪枝（pre-pruning）和后剪枝（post-pruning）。在剪枝前，我们将总样本划分为训练集和验证集。

1）预剪枝

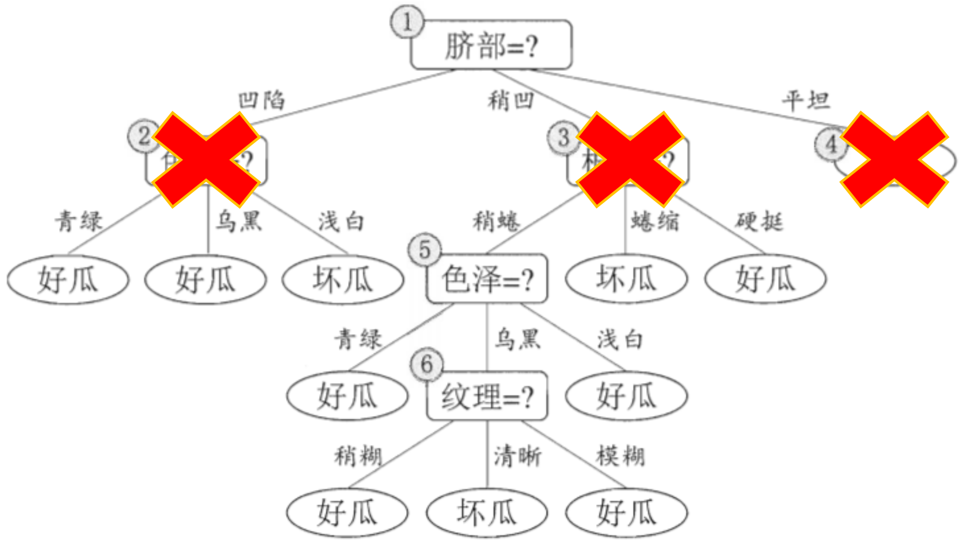
假设未剪枝前的决策树为



我们首先考虑①脐部=？在划分前和划分后的精度（用验证集来测试），发现划分后精度提升，于是划分为凹陷、稍凹、平坦三种情况。

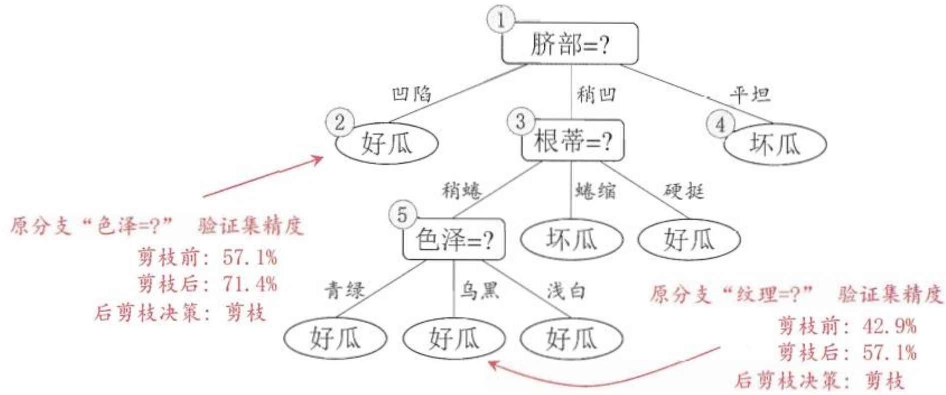


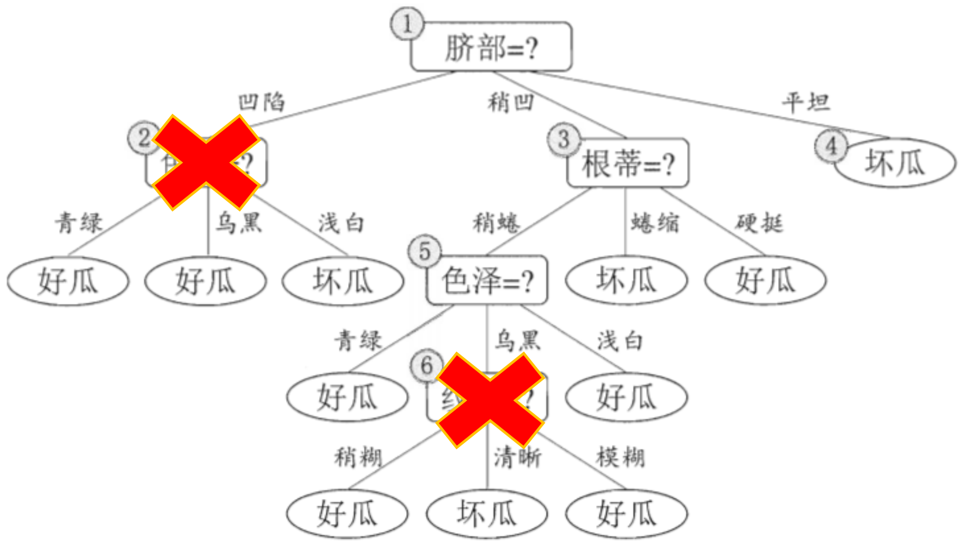
然后考虑②③④，发现再划分后精度无法提高，于是不须继续划分，我们得到了简化的决策树，模型复杂度下降。



2）后剪枝

与预剪枝恰恰相反，后剪枝从末端开始检查精度，直到精度无法提升为止。





一般情形下，后剪枝决策树通常比预剪枝决策树保留了更多的分支。一般情形下，后剪枝决策树的欠拟合风险很小，泛化性能往往优于预剪枝决策树，但后剪枝过程是在生成完全决策树之后进行的，训练时间开销较大。

2. 连续与缺失值

1）连续值处理

假设有这样 5 个西瓜的数据：

编号	含糖率	好瓜
1	0.5	是
2	0.4	否
3	0.3	是
4	0.2	否
5	0.1	否

含糖率是连续值，如何划分？一种常用的方法是二分法。

我们选取 4 个划分点 0.15$\left(\frac{0.1+0.2}{2}\right)$、0.25$\left(\frac{0.2+0.3}{2}\right)$、0.35、0.45

此时$\mathrm{Ent}\left(总样本\right) = -\frac{2}{5}\log_2\frac{2}{5} - \frac{3}{5}\log_2\frac{3}{5} = 0.97$

① 考虑密度⩽0.15 情况下的信息熵，此时有 1 个西瓜（否） $\mathrm{Ent}\left(密度 \leq 0.15\right) = 0$

② 考虑密度⩾0.15 情况下的信息熵，此时有 4 个西瓜，其中 2 个好瓜，2 个坏瓜

$\mathrm{Ent}\left(密度 \geq 0.15\right) = 1$

③ 综合①②可得 $\mathrm{Ent}\left(密度\right) = \frac{1}{5} \times 0 + \frac{4}{5} \times 1 = 0.8$ Gain（总样本，密度）=0.97-0.8=0.17

④ 分别考虑选取 0.25/0.35/0.45 作为划分点的情况，重复①②③的计算过程，比较信息增益。信息增益最大的点作为划为点。

2）缺失值处理

① 当缺失值影响不大时，可以考虑剔除

② 当缺失值影响较大时，考虑使用以下方法

假设有这样 5 个西瓜的数据：

编号	色泽	敲声	好瓜
1	乌黑	沉闷	是
2	乌黑	浊响	否
3	青绿	-	是
4	青绿	-	是
5	-	浊响	是

我们先来看“色泽”的信息增益计算

A 总样本的信息熵

$$\mathrm{Ent}\left(\widetilde{总样本}\right)=-\frac{1}{4}\log_2\frac{1}{4}-\frac{3}{4}\log_2\frac{3}{4}=0.81$$

B 子集的信息熵

$$\mathrm{Ent}\left(\widetilde{色泽=乌黑}\right)=-\frac{1}{2}\log_2\frac{1}{2}-\frac{1}{2}\log_2\frac{1}{2}=1$$

$$\mathrm{Ent}\left(\widetilde{色泽=青绿}\right)=-\frac{2}{2}\log_2\frac{2}{2}-0\log_2 0=0$$

C 信息增益

$$\mathrm{Gain}\left(\widetilde{总样本}，\widetilde{色泽}\right)=0.81-\frac{1}{2}\times 1-0=0.31$$

$$\mathrm{Gain}\left(总样本，色泽\right)=\frac{4}{5}\times 0.31=0.25$$

以同样的方式计算敲声的信息增益

A 总样本的信息熵

$$\mathrm{Ent}\left(\widetilde{总样本}\right)=-\frac{1}{3}\log_2\frac{1}{3}-\frac{2}{3}\log_2\frac{2}{3}=0.92$$

B 子集的信息熵

$$\text{Ent}\left(\widetilde{\textit{敲声} = \textit{沉闷}}\right) = 0$$

$$\text{Ent}\left(\widetilde{\textit{敲声} = \textit{浊响}}\right) = 1$$

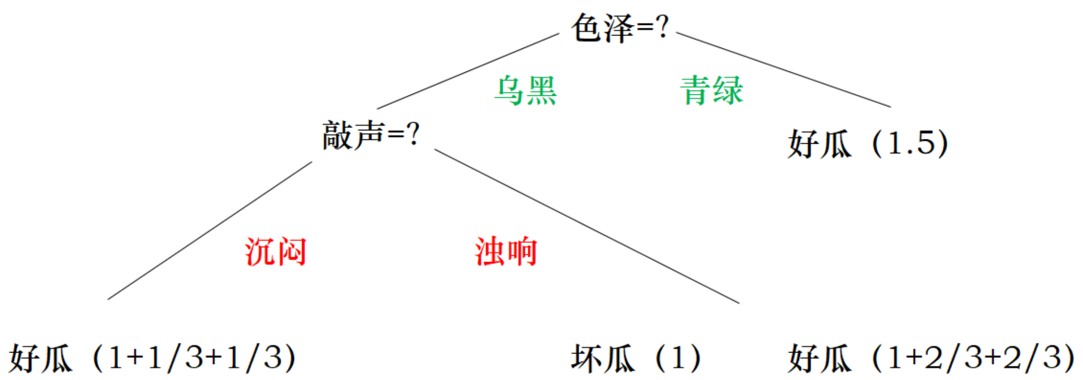
C 信息增益

$$\text{Gain}\left(\widetilde{\text{总样本}}，\widetilde{\text{敲声}}\right) = 0.92 - \frac{2}{3} \times 1 - 0 = 0.25$$

$$\text{Gain}\left(\text{总样本}，\text{敲声}\right) = \frac{3}{5} \times 0.25 = 0.15$$

直观上可以简单认为：给定一个实例，若以色泽进行划分，该实例有 4/5 的概率表现出非缺失值的子集特征，这种不确定性会削减信息增益。

我们绘出决策树：



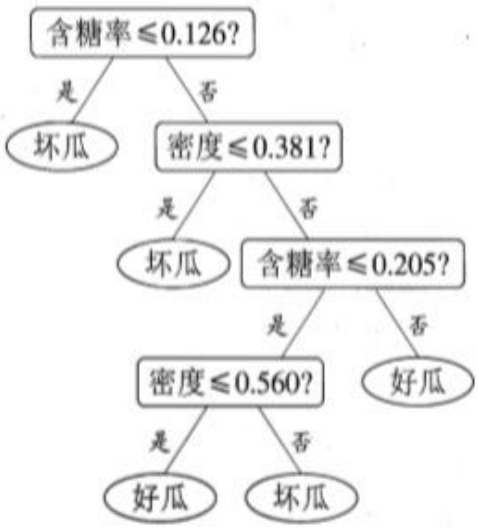
注意的是，括号里代表是权重。比如［色泽=乌黑，敲声=浊响］的情况下，是好瓜的概率= $\dfrac{\textit{好瓜的权重}}{\textit{好瓜的权重}+\textit{坏瓜的权重}}$ 那么，为什么这里好瓜的权重是 1+2/3+2/3？因为 敲声属性缺失的两个样本是好瓜，所以这两个好瓜分别以 1/3、2/3 的权重分配入【敲声=沉闷】、【敲声=浊响】两个分支中。

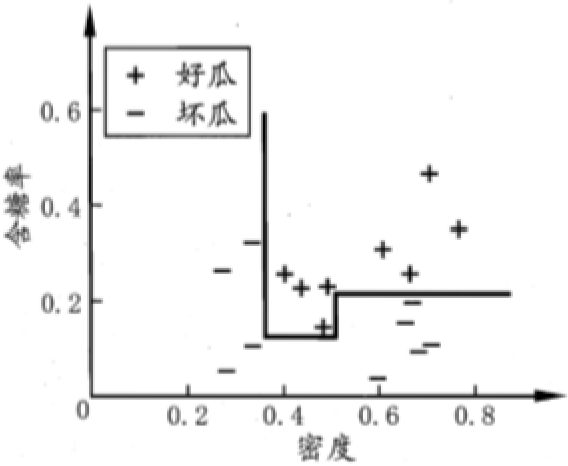
3．多变量决策树

给定一个数据集（来自周志华《机器学习》）

编号	密度	含糖率	好瓜
1	0.697	0.460	是
2	0.774	0.376	是
3	0.634	0.264	是
4	0.608	0.318	是
5	0.556	0.215	是
6	0.403	0.237	是
7	0.481	0.149	是
8	0.437	0.211	是
9	0.666	0.091	否
10	0.243	0.267	否
11	0.245	0.057	否
12	0.343	0.099	否
13	0.639	0.161	否
14	0.657	0.198	否
15	0.360	0.370	否
16	0.593	0.042	否
17	0.719	0.103	否

为了更好拟合它，我们作出下列决策树





可见决策树复杂，且容易过拟合。我们试图

建立一个线性分类器解决这个问题：