人工智能实验

决策树

第 2 次实验

姓名：陈家豪

班级：2018 级计科 1 班

学号：18308013

目录

1 算法原理

决策树是一种经典的分类算法，其最主要的思想就是根据样本的属性，通过一系列顺序规则对样本进行分类、决策。这一系列的规则可以使用树来形象表示，即所谓的决策树。

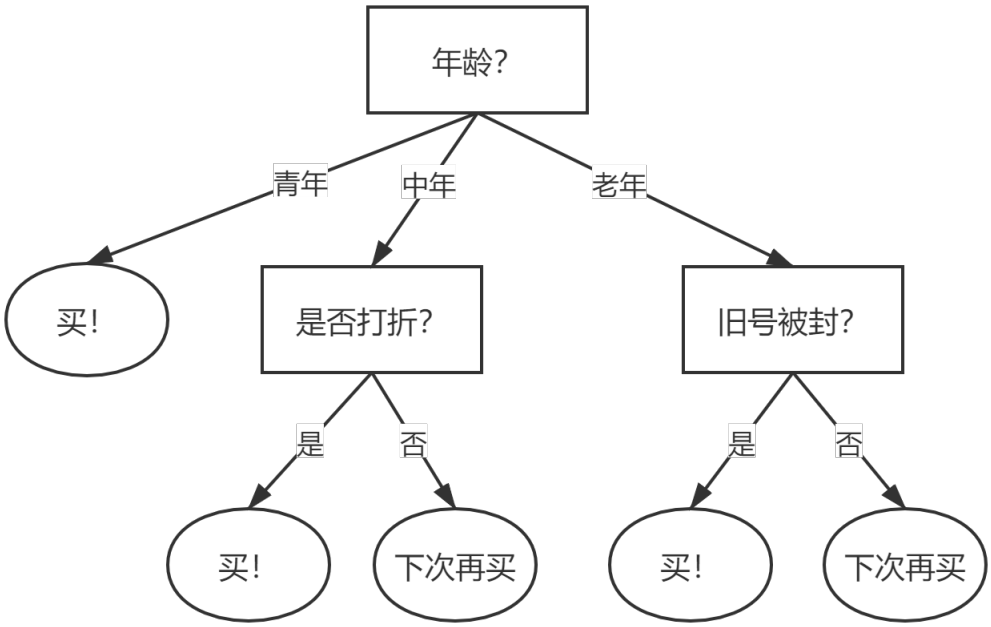

图 1: 决策树：是谁在买 GTA5?

决策树算法的建树步骤如下：

(1) 初始化：创建根节点，其拥有全部数据集和全部特征；

(2) 选择特征：遍历当前结点的数据集和特征，根据某种原则，选择一个特征；

(3) 划分数据；

(4) 创建结点：为每个子数据集创建一个子节点，并删去刚刚选中的特征；

(5) 递归建树：对每个子节点，回到步骤 (2)，重复 (2) 至 (4)，直到达到边界条件，回溯；

(6) 完成建树：叶结点采用多数投票的方式判定自身的类别。

接下来，我们以本次实验中所使用的数据集 car_train.csv 为例，介绍算法具体原理与建树过程。

序号	buying	maint	dorrs	persons	lug_boot	safety	Label
1	high	vhigh	3	2	small	high	0
2	high	med	3	2	small	low	0
3	vhigh	low	2	4	big	low	0
4	med	high	4	2	big	med	0
5	vhigh	med	4	4	med	high	1
6	med	high	4	2	small	low	0
7	low	med	5more	4	med	med	1
8	med	med	2	2	med	low	0
9	vhigh	high	4	more	big	high	0
10	high	low	3	more	small	med	0
...	...	...	...	...	...	...	...

表 1: 数据集 car_train.csv（部分）

1.1 节点特征属性的选择

在构建决策树时，最重要的步骤是为当前建立的节点选择一个合适的属性。正如图1所示，我们选择年龄作为根节点时，可以导出三个分支，各自对应下一个属性节点/叶节点。一个好的决策树，需要在一开始就对数据进行良好的划分，从而实现搜索路径最短的同时有良好的预测效果。为了衡量划分结果的好坏，我们需要引入信息熵的概念。

1.1.1 信息熵

决策树的构造过程可以理解为寻找纯净划分的过程。一个划分越纯净（纯度越高），那么划分后的集合内各样本的目标变量分歧越小。

我们可以用信息熵来近似这种纯度的概念。在信息论中，信息熵用于衡量信息的不确定性。信息熵越大，其包含的信息量就越大，信息的不确定度也就越大。信息熵计算公式如下所示：

$$Entropy(D_t) = -\sum_{i=0}^{c-1} p(i|D_t) \log_2 p(i|D_t) \tag{1}$$

$Entropy(D_t)$ 是节点 D_t 的信息熵。节点 D_t 共有 c 种分类，$p(i|D_t)$ 代表了节点 D_t 为分类 i 的概率。

例如，对于表1，当属性 $dorrs$ 为 4 时，有 1 个样本的 $level$ 为 1，3 个样本为 0，则：

$$Entropy(D_{dorrs=4}) = -\frac{1}{4}\log_2\frac{1}{4} - \frac{3}{4}\log_2\frac{3}{4} = 0.8113$$

我们可以认为，一个集合的信息熵越大，其纯度就越低。我们在构建决策树时，就是不断选择划分的特征参数，将数据集合进行不断划分，得到纯度越来越高的子集合，这样的话在预测子集合的目标变量时就可以有更高的准确率。那么如何衡量划分的效果，即某个特征的划分是否可以得到纯度比原集合更高的子集合呢？我们有三种决策方式：信息增益、信息增益率和 GINI 系数。

1.1.2 信息增益

ID3 算法使用信息增益作为决策策略，以此衡量一个划分效果的好坏。

一次良好的划分可以带来纯度的提高和信息熵的下降，而信息增益的计算就是父亲节点的信息熵减去所有子节点的归一化信息熵。具体计算公式如下所示：

$$Gain(D,a) = Entropy(D) - \sum_{i=1}^{k} \frac{|D_i|}{|D|} Entropy(D_i) \tag{2}$$

其中 D 为父亲节点。D 根据方式 a 对数据集进行划分，得到 k 个子数据集，即子节点 D_i。$|D|$ 和 $|D_i|$ 分别为父亲节点和子节点的样本数量。

以表1为例，假设序号 1 至 10 即为父亲节点 D 的数据集，我们依据特征 safety 进行划分，则可以得到 D_{low}，D_{med} 和 D_{high}，那么

$$\begin{aligned} Gain(D, safety) &= Entropy(D) - \sum_{i} \frac{|D_i|}{|D|} Entropy(D_i) \\ &= 0.7219 - \frac{4}{10} \times 0 - \frac{3}{10} \times 0.9183 - \frac{3}{10} \times 0.9183 \\ &= 0.1709 \end{aligned}$$

从信息增益的定义可见，一次划分的信息增益越大时，说明划分效果越好。因此，在选取节点特征属性时，我们可以计算当前所有特征的信息增益，然后选取其中信息增益最大的特征属性，即为当前节点特征属性。

1.1.3 信息增益率

ID3 算法的信息增益简单有效，但存在一些缺陷，其中之一就是倾向于选择取值多的属性。为了避免这个问题，C4.5 算法采用信息增益率来衡量划分的好坏、以此选择属性。

信息增益率的计算公式如下所示：

$$GainRatio(D,a) = \frac{Gain(D,a)}{SplitInfo(D,a)} = \frac{Gain(D,a)}{-\sum_{j=1}^{v} \frac{|D_j|}{|D|} \log_2 \frac{|D_j|}{|D|}} \tag{3}$$

其中 $SplitInfo(D,a)$ 为 D 关于特征 a 的熵。D 使用方式 a 划分成 v 个子数据集，即为 D_j。

同样，当一次划分的信息增益率越大时，划分的效果越好。在选取节点特征属性时，我们可以计算当前所有特征的信息增益率，然后选取其中信息增益率最大的特征属性，即为当前节点特征属性。

1.1.4 GINI 系数

除了信息增益和信息增益率，我们还有第三种方法衡量划分的好坏，即 CART 决策树所采用的 GINI 系数。GINI 系数计算公式如下：

$$gini(D,a) = \sum_{j=1}^{k} p(a_j) \times gini(D_j|a=a_j) = \sum_{j=1}^{k} (\frac{|D_j|}{|D|} \times \sum_{i=1}^{n} p_i(1-p_i)) \tag{4}$$

其中 $gini(D,a)$ 表示 D 在方式 a 的划分下的 GINI 系数，D_j 是 D 经过方式 a 划分后的子数据集。p_i 表示在 D_j 中不同目标变量值的比例。

以表1为例，假设我们依据 lug_boot 对前十个样本进行划分，则 GINI 系数为：

$$\begin{aligned} gini(D, lug\_boot) &= p(a_{small}) \times gini(D_{small}) + p(a_{med}) \times gini(D_{med}) + p(a_{big}) \times gini(D_{big}) \\ &= \frac{4}{10} \times (1 - 1^2 - 0^2) + \frac{3}{10} \times (1 - (\frac{2}{3})^2 - (\frac{1}{3})^2) + \frac{3}{10} \times (1 - 1^2 - 0^2) \\ &= 0.1333 \end{aligned}$$

与信息增益、信息增益率不同的是，当一次划分的 GINI 系数越小时，划分效果越好。此外，CART 算法构建的是二叉树，这意味着 D 经过划分后只能有两个子数据集，即 $k = 2$。因此，当特征属性有多种取值时，我们要将它们合并变成两种取值。

1.2 决策树的构建

基于1.1所介绍的信息熵、信息增益、信息增益率和基尼系数，我们就可以从根节点开始构建一个决策树。我们有三种不同类型的决策树：ID3 决策树、C4.5 决策树和 CART 决策树。

1.2.1 ID3 决策树

ID3 决策树在构造节点时采用信息增益衡量划分效果。还是以表1为例，假设我们已经选择 dorrs 作为上层节点，接下来我们要构造 dorrs=4 下的节点。

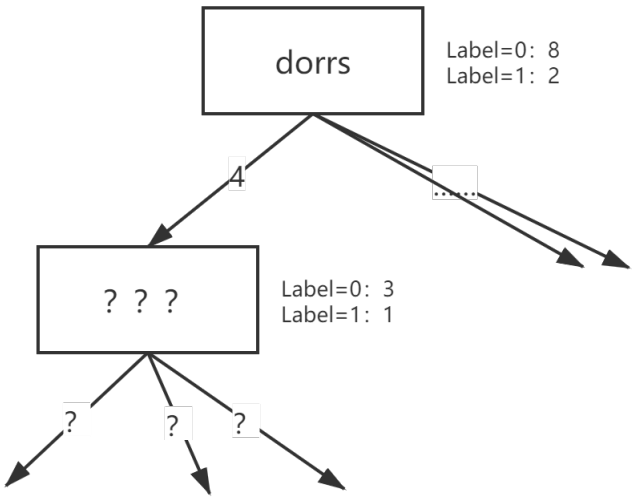

图 2: 构造 dorrs=4 后的分支

序号	buying	maint	persons	lug_boot	safety	Label
4	med	high	2	big	med	0
5	vhigh	med	4	med	high	1
6	med	high	2	small	low	0
9	vhigh	high	more	big	high	0

表 2: dorrs=4 下的子数据集

此时，该节点所拥有的数据集如表2所示。我们计算剩余的各个特征属性的信息增益。信息增益的计算公式见1.1.2的式2，计算结果如下：

$$Gain(D, buying) = 0.3112$$
$$Gain(D, maint) = 0.8113$$
$$Gain(D, persons) = 0.8113$$
$$Gain(D, lug\_boot) = 0.8113$$
$$Gain(D, safety) = 0.3113$$

可见，在选择属性 maint，persons 和 lug_boot 时可以得到最大的信息增益。我们不妨取 maint 作为该节点的特征属性，那么可以构建如图3所示的决策树：

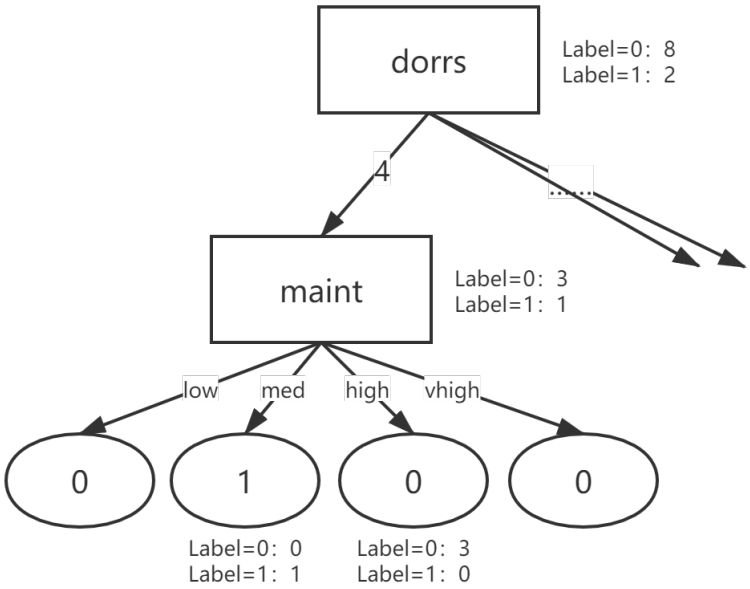

图 3: 选取 maint 作为节点特征属性

当 maint=med 和 main=high 时，均有样本对应，且 Label 一致，故我们不再进一步地划分，而是直接构建相应值的叶节点。当 maint=low 和 maint=vhigh 时，没有样本对应，这时我们选取父节点的众数进行赋值，故叶节点的值均为 1。

以上就是 ID3 决策树构建一个节点的过程。我们从根节点开始递归构建，即可构建整个决策树。

1.2.2 C4.5 决策树

C4.5 决策树的构建与 ID3 相似，区别在于使用信息增益率而不是信息增益选择特征属性。

同样，基于表1的数据集，假设我们已经选择 dorrs 作为根节点，接下来构造 dorrs=4 下的节点，此时决策树如图2所示，子数据集如表2所示。信息增益率的计算公式见1.1.3的式3，则各个特征属性的信息

增益率如下：

$$
\begin{aligned}
GainRatio(D, buying) &= 0.3112 \\
GainRatio(D, maint) &= 1 \\
GainRatio(D, persons) &= 0.5409 \\
GainRatio(D, lug\_boot) &= 0.5409 \\
GainRatio(D, safety) &= 0.2075
\end{aligned}
$$

可见，选择特征属性 maint 可以获得最大的信息增益率，故我们选择 maint 作为该节点的特征属性。同样，该节点的子节点均无法继续划分，故均为叶节点。最终如图3所示。

1.2.3 CART 决策树

CART 决策树采用基尼系数进行特征选择，但与 ID3 决策树、C4.5 决策树不同的是，CART 决策树是一个二叉树。这意味着，当我们选取某个特征属性作为当前决策节点时，如果该特征属性有多于两种的取值，我们要对取值进行合并。

假设当前有特征属性 $A_1, A_2, \cdots, A_n$，对于特征属性 A_i，其取值有 $a_{i,1}, a_{i,2}, \cdots, a_{i,n_i}$。那么对每个特征属性 A_i，对其每个取值 $a_{i,j}$，根据样本对 $A_i = a_{i,j}$ 的测试为是或否，将数据集 D 分为 D_1 和 D_2，可以计算基尼指数 $gini(D, A_i = a_{i,j})$。最终，我们获得所有特征属性及其对应取值的基尼指数，从中选取指数最小的特征属性及其取值，即可构建当前节点，数据集以此划分为两个子数据集，分别参与两个子节点的计算。

以表1的数据集为例，假设我们已经选择 dorrs 作为根节点，接下来构造 dorrs=4 下的节点，此时决策树如图2所示，子数据集如表2所示。那么对于特征属性 persons，其各个取值的基尼指数分别为：

$$
\begin{aligned}
gini(D, persons = 2) &= \frac{|D_{p=2}|}{|D|} \times (1 - 1^2 - 0^2) + \frac{|D_{p\neq 2}|}{|D|} \times (1 - (\frac{1}{2})^2 - (\frac{1}{2})^2) \\
&= 0.25 \\
gini(D, persons = 4) &= \frac{|D_{p=4}|}{|D|} \times (1 - 1^2 - 0^2) + \frac{|D_{p\neq 4}|}{|D|} \times (1 - 1^2 - 0^2) \\
&= 0 \\
gini(D, persons = more) &= \frac{|D_{p=more}|}{|D|} \times (1 - 1^2 - 0^2) + \frac{|D_{p\neq more}|}{|D|} \times (1 - (\frac{1}{3})^2 - (\frac{2}{3})^2) \\
&= 0.33
\end{aligned}
$$

可见，在特征属性 persons 中，选取值 4 作为划分点效果最佳。如果其他特征属性相应取值的基尼指数无法比它更小时，我们便选取 persons=more 作为当前节点的划分。当前数据集可以根据样本的 persons 是否为 4 划分成两个子数据集，各自参与下一个子节点的计算。同样，这两个数据集内样本的标签是一致的，故我们不继续计算，而是直接设立叶结点。最终决策树如图4所示。

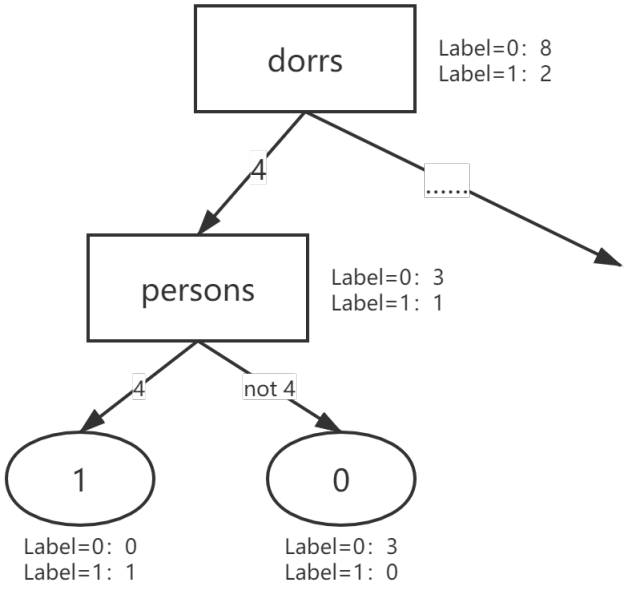

图 4: 选取 persons=4 作为节点特征属性

1.2.4 边界条件

从上述讨论可知，决策树的构造就是不断地递归构造节点，直到数据集不能或不需继续划分为止。假设当前节点的数据集为 D，这个递归边界条件如下：

(1) D 中的所有样本属于同一个类别 C，则当前节点标记为 C 类叶节点；

(2) D 的全部特征属性已确定，即 D 的特征属性取值均一致时，此时无法继续划分，则将当前节点标记为叶节点，类别为 D 中出现最多的类别；

(3) D 为空集，即不存在样本满足当前分支的特征属性时，则将当前节点标记为叶节点，类别为父结点中出现最多的类。

1.2.5 剪枝

当决策树过于复杂时，容易出现过拟合情况。我们需要对决策树进行剪枝以提升泛化性能。剪枝有两种方式，分别是预剪枝和后剪枝。

预剪枝在决策树的生成过程中进行，实时判断当前的节点是否应当继续划分。如果划分前后，决策树在验证集上的准确率不提高，则进行剪枝、无需划分。

后剪枝是在生成完整的决策树后自底向上考察非叶节点。如果对于某个非叶节点，在转变成叶节点后决策树在验证集上的准确率没有降低，则将其变成叶节点。

1.2.6 连续型数据的处理

在上述原理描述中，我们一直是基于离散型数据介绍决策树算法。当数据为连续型时该如何处理呢？我们可以将连续型数据进行离散化处理，比如属于某个取值范围的数据划为一类，属于另一个取值范围的划为另一类，从而实现连续型数据的离散化。

2 伪代码/流程图

2.1 ID3 决策树

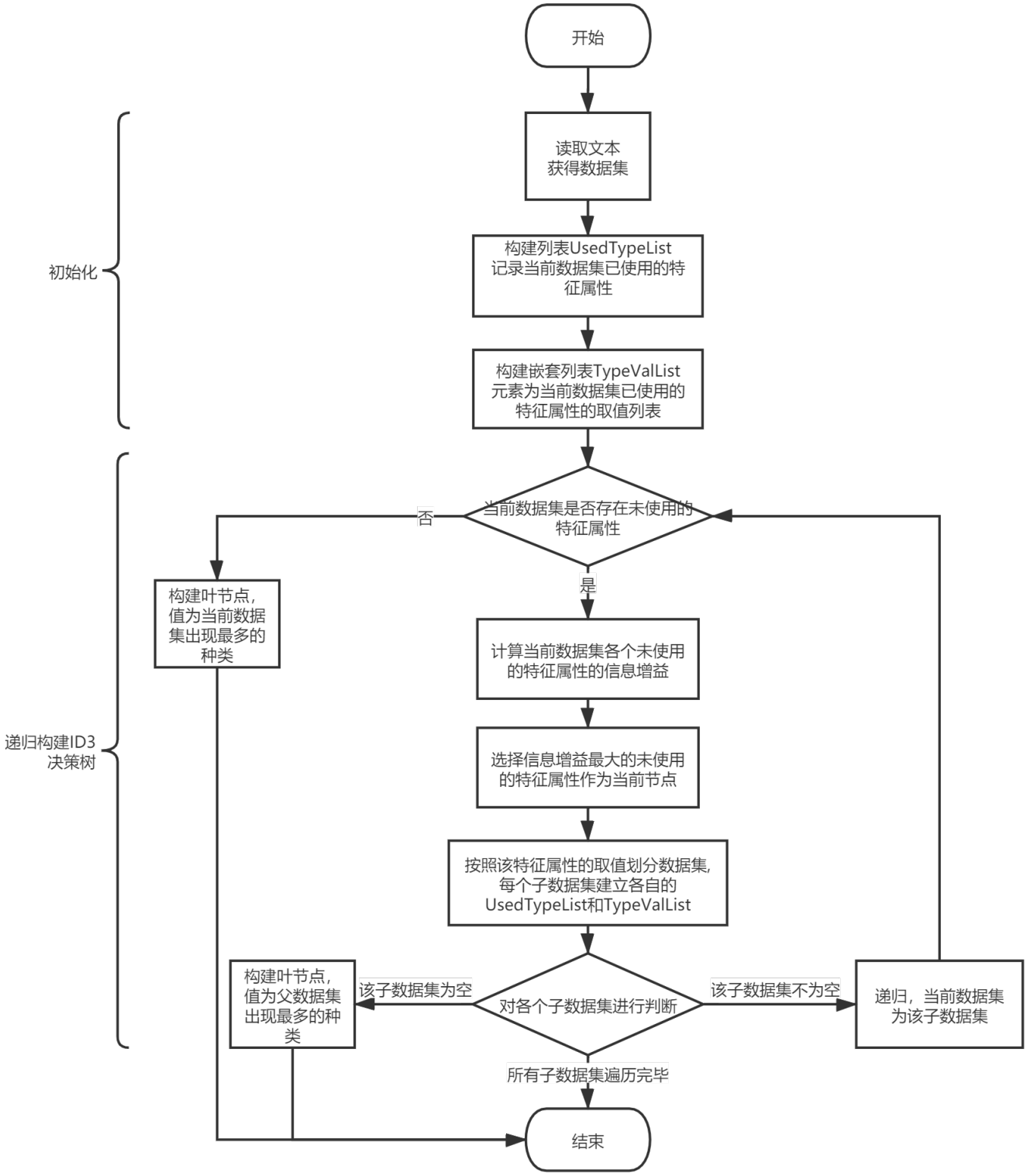

图 5: 构建 ID3 决策树流程图

2.2 C4.5 决策树

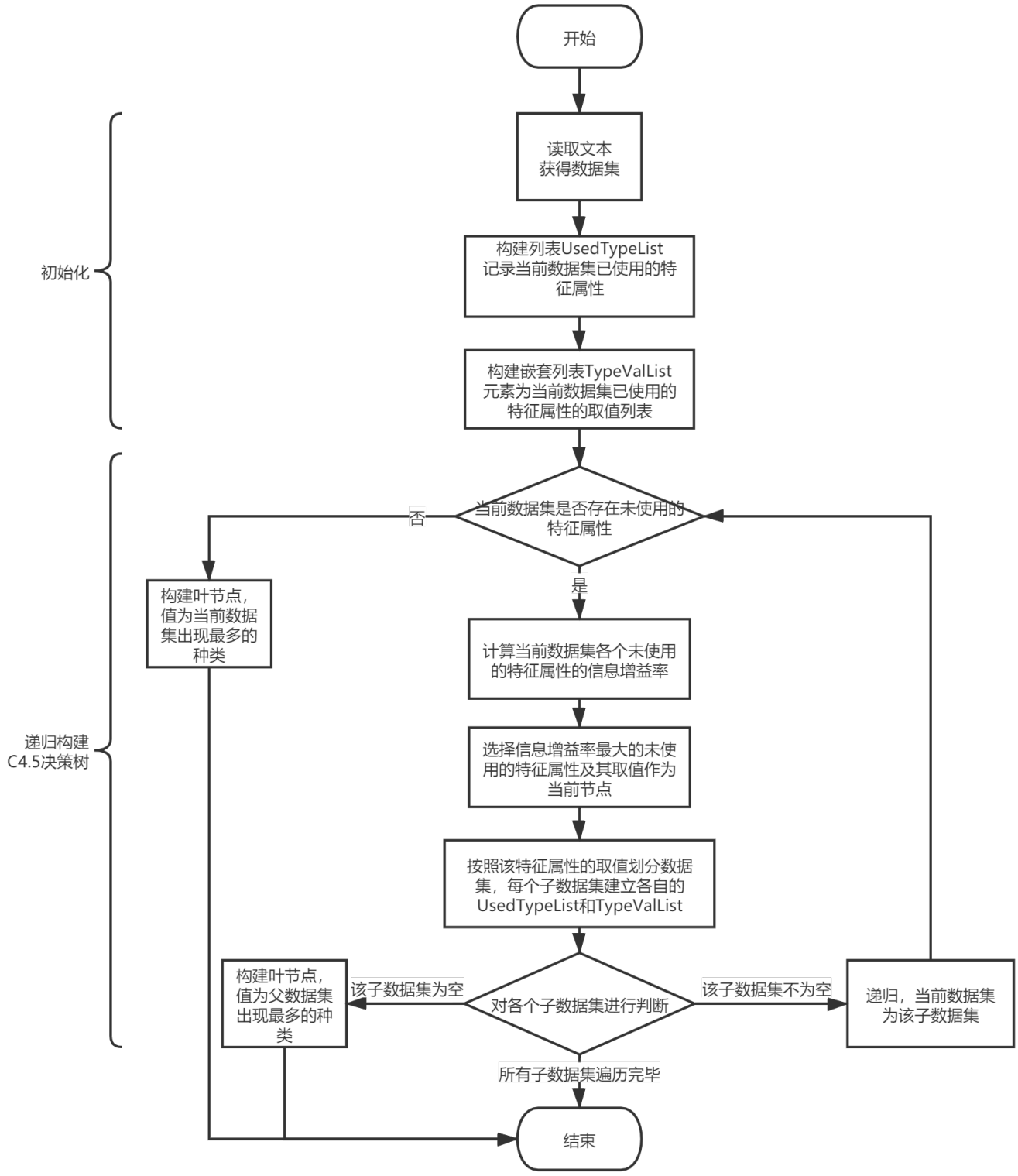

图 6: 构建 C4.5 决策树流程图

2.3 CART 决策树

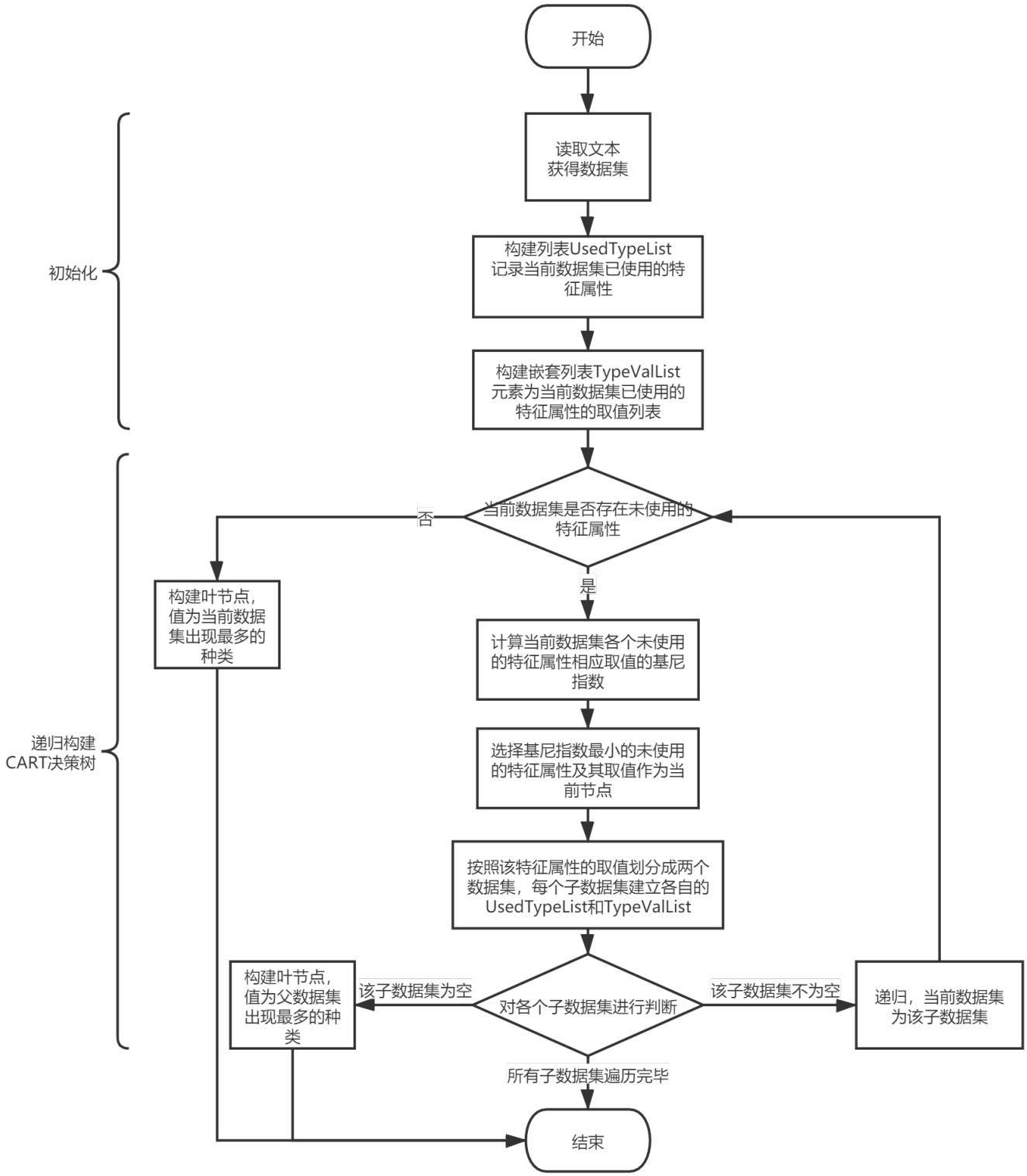

图 7: 构建 CART 决策树流程图

3 代码截图

3.1 信息熵的计算

计算信息熵的代码如图8所示。

```python
def Calculate_Entropy(DataLists,TypeUsedList,TypeValList):
    """
    计算信息熵
    """
    pos_num=0                    #当前数据集中样本标签为1的数量
    total_num=0                  #当前数据集样本数量
    for DataList in DataLists:
        #判断当前样本是否位于当前数据集
        isSatisfy=True
        for i in range(len(TypeUsedList)):
            if TypeUsedList[i]==True:
                if DataList[i] in TypeValList[i]:
                    continue
                else:
                    isSatisfy=False
                    break
        if isSatisfy:#如果当前样本位于数据集
            total_num+=1
            if DataList[-1]=='1':
                pos_num+=1
    p_pos=pos_num/total_num
    p_neg=(total_num-pos_num)/total_num
    if p_pos!=0 and p_neg!=0:
        return -p_pos*math.log2(p_pos)-p_neg*math.log2(p_neg)
    else:#如果数据集标签全为1或0，信息熵为0
        return 0
```

图 8: 计算信息熵函数 Calculate_Entropy()

其中，DataLists 是全部数据集列表，元素为各个样本的属性值列表。TypeUsedList 和 TypeValList 用于从 DataLists 中划分出当前节点的数据集。TypeUsedList 是一个 bool 值列表，记录当前数据集已用于划分的特征属性，TypeValList 记录当前数据集已使用的特征属性的决策值。该函数最终返回当前数据集的信息熵。

3.2 信息增益的计算

计算信息增益的代码如图9所示。

```python
def Calculate_Gain(DataLists,TypeUsedList,TypeValList,FatherTypeIndex):
    """
    计算信息增益
    """
    FatherValDict=dict()        #字典{特征属性各取值:对应样本数量}
    total_num=0                 #当前数据集样本数量
    for DataList in DataLists:
        #判断当前样本是否位于当前数据集
        isSatisfy=True
        for i in range(len(TypeUsedList)):
            if TypeUsedList[i]==True:
                if DataList[i] in TypeValList[i]:
                    continue
                else:#如果该属性不符
                    isSatisfy=False
                    break
        if isSatisfy:#如果当前样本位于数据集
            total_num+=1
            #统计属于暂定特征属性相应取值的样本数量
            if DataList[FatherTypeIndex] not in FatherValDict.keys():
                FatherValDict[DataList[FatherTypeIndex]]=1
            else:
                FatherValDict[DataList[FatherTypeIndex]]+=1
    #计算当前信息熵
    result=Calculate_Entropy(DataLists, TypeUsedList, TypeValList)
    SonTypeUsedList=copy.deepcopy(TypeUsedList)
    SonTypeUsedList[FatherTypeIndex]=True
    for val in FatherValDict.keys():
        SonTypeValList=copy.deepcopy(TypeValList)
        SonTypeValList[FatherTypeIndex].append(val)
        #计算各个子数据集的信息熵
        temp=Calculate_Entropy(DataLists, SonTypeUsedList, SonTypeValList)
        result-=temp*FatherValDict[val]/total_num
    return result
```

图 9: 计算信息增益函数 Calculate_Gain()

其中，DataLists、TypeUsedList 和 TypeValList 的含义和结构与3.1相同。额外输入参数 FatherTypeIndex 是当前节点假设使用的特征属性在属性列表中的索引值。该函数最终返回当前数据集在暂定特征属性下的信息增益。

3.3 信息增益率的计算

计算信息增益率的代码如图10所示。

```python
def Calculate_GainRatio(DataLists,TypeUsedList,TypeValList,FatherTypeIndex):
    """
    计算信息增益率
    """
    total_num=0                 #当前数据集样本数量
    TypeValDict=dict()          #字典{特征属性各取值:对应样本数量}
    for datalist in DataLists:
        #判断当前样本是否位于当前数据集
        isSatisfy=True
        for i in range(len(TypeUsedList)):
            if TypeUsedList[i]==True:
                if datalist[i] in TypeValList[i]:
                    continue
                else:
                    isSatisfy=False
                    break
        if isSatisfy:#如果属于当前数据集
            total_num+=1
            #统计属于暂定特征属性相应取值的样本数量
            if datalist[FatherTypeIndex] not in TypeValDict.keys():
                TypeValDict[FatherTypeIndex]=1
            else:
                TypeValDict[FatherTypeIndex]+=1
    #计算得到暂定特征属性的信息增益
    Gain=Calculate_Gain(DataLists,TypeUsedList,TypeValList,FatherTypeIndex)
    #计算数据集关于暂定特征的熵
    SplitInfo=0
    for val in TypeValDict.keys():
        if TypeValDict[val]!=0:
            SplitInfo-=(TypeValDict[val]/total_num)*
math.log2(TypeValDict[val]/total_num)
    return Gain/SplitInfo
```

图 10: 计算信息增益率函数 Calculate_GainRatio()

DataLists、TypeUsedList，TypeValList 和 FatherTypeIndex 的含义和结构与3.2相同。

3.4 基尼指数的计算

计算基尼指数的代码如图11所示。

```python
def Calculate_GINI(DataLists,TypeUsedList,TypeValList,FatherTypeIndex, ChosenVal):
    """
    计算基尼指数
    """
    total_num=0                 #当前数据集样本数量
    NumList=[[0,0],[0,0]]       #划分后两个子数据集的正负样本数量
    for datalist in DataLists:
        isSatisfy=True
        #判断当前样本是否位于当前数据集
        for i in range(len(TypeUsedList)):
            if TypeUsedList[i]==True:
                if datalist[i] in TypeValList[i]:
                    continue
                else:
                    isSatisfy=False
                    break
        if isSatisfy:#如果属于当前数据集
            total_num+=1
            if datalist[FatherTypeIndex]==ChosenVal:#如果特征属性取值是预设值
                if datalist[-1]=='1':
                    NumList[0][0]+=1
                else:
                    NumList[0][1]+=1
            else:
                if datalist[-1]=='1':
                    NumList[1][0]+=1
                else:
                    NumList[1][1]+=1
    #计算基尼指数
    gini=0
    for i in range(2):
        temp=NumList[i][0]+NumList[i][1]
        if temp==0:
            gini+=0
        else:
            gini+=(temp/total_num)*(1-pow(NumList[i][0]/temp,2)-pow(NumList[i][1]/temp,2))
    return gini
```

图 11: 计算基尼指数函数 Calculate_GINI()

DataLists、TypeUsedList，TypeValList 和 FatherTypeIndex 的含义和结构与3.2相同。额外的输入参数 ChosenVal 是暂定特征属性的划分点（即是 ChosenVal 的归为一类，不是的归为另一类）。

3.5 ID3 决策树的构建

构建 ID3 决策树的代码如图12所示。

```
def Build_Tree_ID3(DataLists, OriTypeValList, TypeUsedList, TypeValList, Depth):
    """
    构建ID3决策树（基于当前节点）
    """
    total_num=0                 #当前节点样本总数量
    pos_num=0                   #当前节点样本值为1的数量
    for DataList in DataLists:
        #以下检测样本DataList是否属于当前节点
        isSatisfy=True
        for i in range(len(TypeUsedList)):
            if TypeUsedList[i]==True:
                if DataList[i] in TypeValList[i]:
                    continue
                else:
                    isSatisfy=False
                    break
        if isSatisfy:           #如果DataList属于当前节点
            total_num+=1
            if DataList[-1]=='1':#如果样本值为1
                pos_num+=1

    if Depth==len(TypeUsedList): #如果递归达到最深，即所有属性均已决策
        if 2*pos_num>=total_num:
            return 1#返回叶节点1
        else:
            return 0#返回叶节点0
    elif pos_num==0:            #如果样本值均为0，返回叶节点0
        return 0
    elif pos_num==total_num:    #如果样本值均为1，返回叶节点1
        return 1
    else:
        TypeGainDict=dict()     #字典{属性：对应信息增益}
        for i in range(len(TypeUsedList)):
            if TypeUsedList[i]:  #若该属性已决策，跳过
                continue
            #计算未决策属性的信息增益
            TypeGainDict[i]=
Calculate_Gain(DataLists, TypeUsedList, TypeValList, i)
        #以下寻找当前信息增益最大的未决策属性
        MaxGain=-1
        NodeIndex=0             #最大信息增益的未决策属性索引
        for i in TypeGainDict.keys():
            if MaxGain<TypeGainDict[i]:
                MaxGain=TypeGainDict[i]
                NodeIndex=i
        TreeDict=dict()         #子决策树
        NodeValList=[]          #即将决策的属性的值列表
        #填充即将决策的属性的出现值列表
        for DataList in DataLists:
            isSatisfy=True
            for i in range(len(TypeUsedList)):
                if TypeUsedList[i]==True:
                    if DataList[i] in TypeValList[i]:
                        continue
                    else:
                        isSatisfy=False
                        break
            if isSatisfy:
                if DataList[NodeIndex] not in NodeValList:
                    NodeValList.append(DataList[NodeIndex])
        #构建子决策树的TypeUsedList
        NewTypeUsedList=copy.deepcopy(TypeUsedList)
        NewTypeUsedList[NodeIndex]=True
        #构建子决策树
        for NodeVal in OriTypeValList[NodeIndex]:
            TreeKEY=str(NodeIndex)+':'+'1'+':'+NodeVal
#键：  "属性序号:是/否:属性值"
            if NodeVal in NodeValList:
#该属性值有训练集样本对应时，递归构建子决策树
                NewTypeValList=copy.deepcopy(TypeValList)
                NewTypeValList[NodeIndex].append(NodeVal)
                TreeDict[TreeKEY]=
Build_Tree_ID3(DataLists, OriTypeValList, NewTypeUsedList, NewTypeValList,Depth
+1)
            else:
#该属性值没有训练集样本对应时，构建叶节点
                if pos_num>=total_num-pos_num:
                    TreeDict[TreeKEY]=1
                else:
                    TreeDict[TreeKEY]=0
        return TreeDict
```

图 12: 构建 ID3 决策树函数 Build_Tree_ID3()

函数的输入参数 DataLists 是全部数据集列表；OriTypeValList 是特征属性取值列表，是一个二重嵌套列表，列表元素为各个特征属性的所有取值组成的列表。TypeValList 也是二重嵌套列表，记录当前数据集已使用的特征属性的取值。Depth 是当前树的深度。

该函数是一个递归函数。在本次实验中，我使用字典来构建树。每个字典就相当于一个节点，键为一句用于检索的字符串，具有固定格式，为 “属性序号: 1/0 : 属性取值”。其中 1 表示该属性是该值时搜索该分支，0 表示该属性不是该值时搜索该分支。值为该键对应的子节点，如果是中间节点，那么子节点是下一个字典；如果是叶节点，那么子节点就是一个整数值，取值为 0 或 1。

调用该函数后，最终返回一个字典，即为 ID3 决策树，我们使用该字典对一个样本数据进行不断搜索，最终得到预测值。

3.6 C4.5 决策树的构建

C4.5 决策树的构建与 ID3 决策树的构建过程极为相似，不同之处仅在于 C4.5 决策树使用信息增益率而不是信息增益来选择当前节点所要采用的特征属性。故此处不再粘贴代码，具体代码详见 “DecisionTree.py” 的 Build_Tree_C45() 函数。

3.7 CART 决策树的构建

CART 决策树相比前两种决策树的不同之处在于两点，一是要构建二叉树，二是要使用基尼指数选择当前节点的特征属性和划分值。具体代码如图13所示：

```
def Build_Tree_CART(DataLists, OriTypeValList, TypeUsedList, TypeValList, Depth):
    """
    构建CART决策树（基于当前节点）
    """
    ......
    ......
    ......
    else:
        TypeGINIDict=dict()      #字典{属性：对应信息增益}
        for i in range(len(TypeUsedList)):
            if TypeUsedList[i]:  #若该属性已决策，跳过
                continue
            #计算未决策属性的信息增益
            TypeGINIDict[i]=[]
            for val in OriTypeValList[i]:
                TypeGINIDict[i].append(Calculate_GINI(DataLists, TypeUsedList, TypeValList, i,val))

        #以下寻找当前最小基尼系数的未决策属性和对应的属性值
        MinGINI=-1
        NodeIndex=0              #最小基尼系数的未决策属性索引
        NodeVal='\0'             #最小基尼系数的未决策属性索引的划分值
        for i in TypeGINIDict.keys():
            for j in range(len(TypeGINIDict[i])):
                if MinGINI==-1 or MinGINI>=TypeGINIDict[i][j]:
                    MinGINI=TypeGINIDict[i][j]
                    NodeIndex=i
                    NodeVal=OriTypeValList[i][j]
        TreeDict=dict()          #子决策树
        isValExist=[False,False]          #即将决策的属性的值列表
        #填充即将决策的属性的出现值列表
        for DataList in DataLists:
            isSatisfy=True
            for i in range(len(TypeUsedList)):
                if TypeUsedList[i]==True:
                    if DataList[i] in TypeValList[i]:
                        continue
                    else:
                        isSatisfy=False
                        break
            if isSatisfy:
                if DataList[NodeIndex]==NodeVal:
                    isValExist[0]=True
                else:
                    isValExist[1]=True
            if isValExist[0] and isValExist[1]:
                break
        #构建子决策树的TypeUsedList
        NewTypeUsedList=copy.deepcopy(TypeUsedList)
        NewTypeUsedList[NodeIndex]=True
        ##构建符合取值的子决策树
        TreeKEY=str(NodeIndex)+':'+'1'+':'+NodeVal
        if isValExist[0]:#如果符合取值的子数据集存在，构造下一个节点
            NewTypeValList=copy.deepcopy(TypeValList)
            NewTypeValList[NodeIndex].append(NodeVal)
            TreeDict[TreeKEY]=Build_Tree_CART(DataLists, OriTypeValList, NewTypeUsedList, NewTypeValList,Depth+1)
        else:#如果符合取值的子数据集不存在，构造叶节点
            if pos_num>total_num-pos_num:
                TreeDict[TreeKEY]=1
            else:
                TreeDict[TreeKEY]=0
        #构建不符合取值的子决策树
        TreeKEY=str(NodeIndex)+':'+'0'+':'+NodeVal
        if isValExist[1]:#如果不符合取值的子数据集存在，构造下一个节点
            NewTypeValList=copy.deepcopy(TypeValList)
            for val in OriTypeValList[NodeIndex]:
                if val!=NodeVal:
                    NewTypeValList[NodeIndex].append(val)
            TreeDict[TreeKEY]=Build_Tree_CART(DataLists, OriTypeValList, NewTypeUsedList, NewTypeValList,Depth+1)
        else:#如果符合取值的子数据集存在，构造叶节点
            if pos_num>total_num-pos_num:
                TreeDict[TreeKEY]=1
            else:
                TreeDict[TreeKEY]=0
        return TreeDict
```

图 13: 构建 CART 决策树函数 Build_Tree_CART()

该函数的输入参数与 Build_Tree_ID3()、Build_Tree_C45() 相同。第 5 至 6 行的省略号省略了与图12中 Build_Tree_ID3() 函数第 5 至 30 行相同的代码。同样，该函数使用字典构建决策树。我们可以随后使用该字典对样本数据进行搜索，得到相应预测结果。

4 实验过程与结果分析

4.1 训练集与验证集的划分

本次实验只提供了一个数据集，要求我们自行划分训练集和验证集。我按照训练集: 验证集 =7:3 的比例划分数据集，前 1210 个样本数据为训练集，后续样本为验证集。然后基于训练集构建三种决策树，并使用验证集检验准确率。

4.2 三种决策树的准确率与分析

三种决策树在验证集上的准确率如表3所示。

决策树	准确率
ID3	0.9633204633204633
C4.5	0.9633204633204633
CART	0.8957528957528957

表 3: 三种决策树的准确率

可见，三种决策树中，ID3 决策树准确率和 C4.5 决策树准确率一致，CART 决策树准确率最低。显然，CART 决策树的准确率与其他决策树相比有着较大差距。如果我们不限定 CART 决策树必须采用二叉树构造，那么准确率可以达到 0.9575%。我认为，这可能是与 CART 决策树在采用二叉树构建实现时，强制将不同取值的分支进行合并，导致了准确率下降。一方面，二叉树的使用可以简化决策树，提高泛化能力；另一方面，二叉树也可能导致欠拟合的情况出现。

此外，在和同学交流过程中，我得知如果在实现 CART 决策树时，可以重复使用特征对数据集进行划分，那么准确率可以达到 99%+。由于时间有限，加上咨询 TA 得知该实现可能存在问题，故我没有进一步验证。

验证集的前五个样本的真实值与预测值如表4所示。可见 ID3 决策树和 C4.5 决策树的预测值与真实值一致，而 CART 预测值在第 4 个验证集样本预测错误。

序号	样本	真实值	ID3 预测值	C4.5 预测值	CART 预测值
1211	low,low,3,2,big,low	0	0	0	0
1212	low,med,5more,more,small,med	1	1	1	1
1213	low,high,2,more,med,high	1	1	1	1
1214	vhigh,high,5more,more,med,med	0	0	0	1
1215	high,med,5more,4,big,low	0	0	0	0

表 4: 前 5 个验证集样本及其真实值、预测值

由于三种算法最终构建出来的决策树均比较复杂，故我在此处提供验证集的前三个数据样本的搜索路径以供参考。如图14、图15和图16所示。

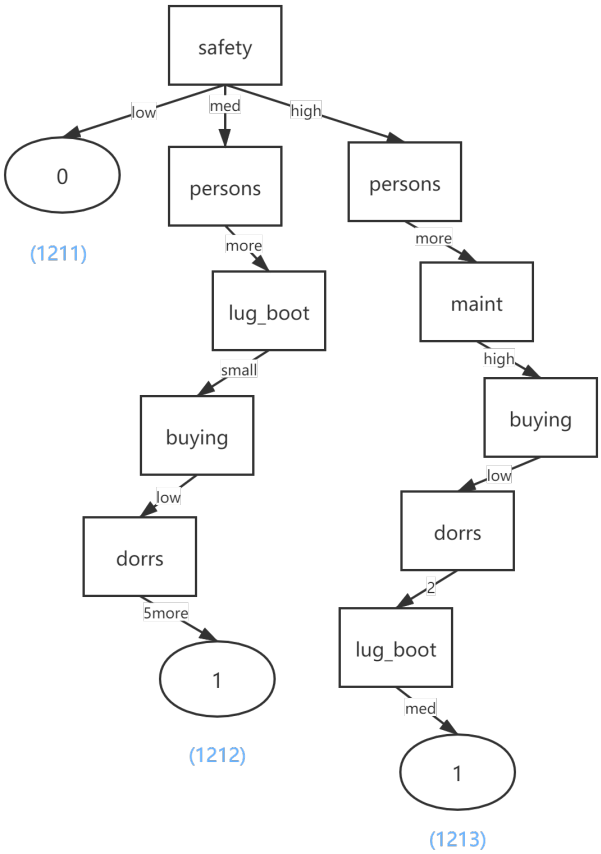

图 14: 在 ID3 决策树上的搜索路径

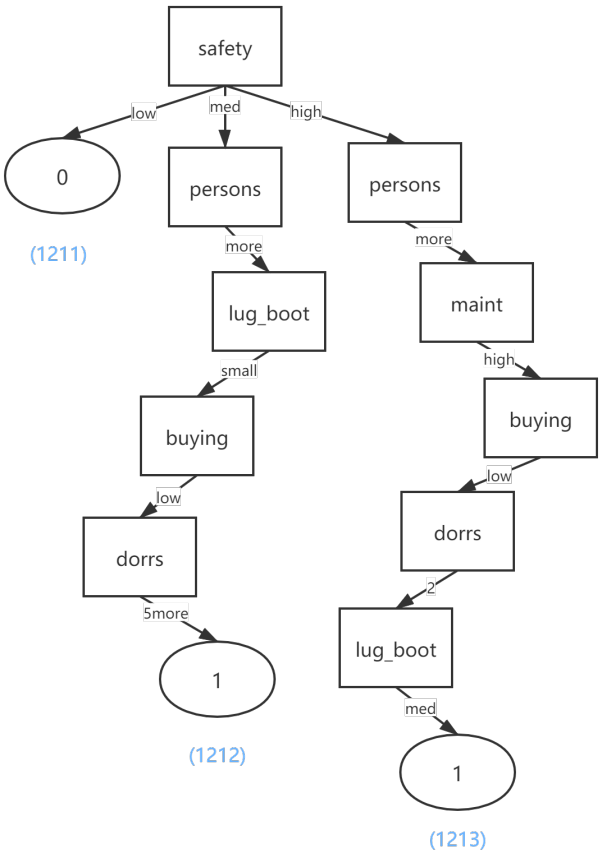

图 15: 在 C4.5 决策树上的搜索路径

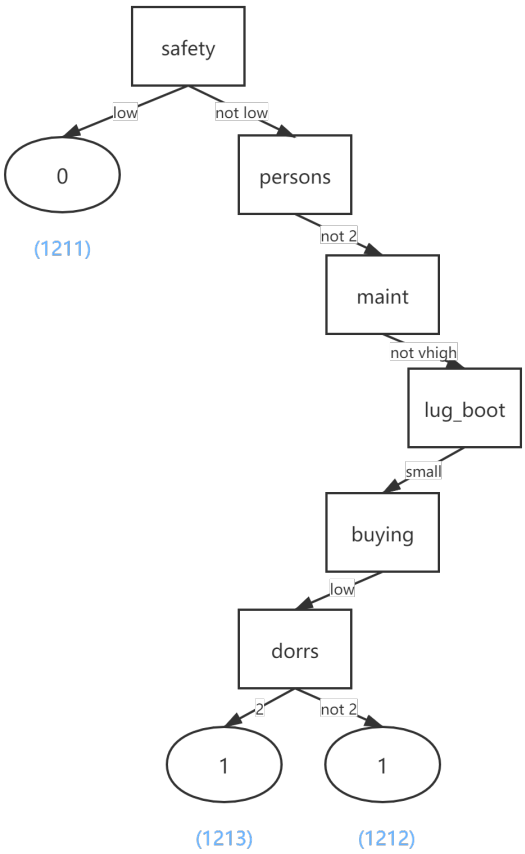

图 16: 在 CART 决策树上的搜索路径

我们可以发现，在 CART 决策树中，第 1212 个和第 1213 个前期的路径一致，最后一个节点值均为 1。显然这里可以进行剪枝处理。

5 思考题

5.1 决策树避免过拟合的方法

决策树的构建非常容易出现过拟合的情况，导致泛化能力降低。我们可以从两方面入手：样本和决策树算法本身。

5.1.1 样本问题

当样本噪声干扰过大，或者引入了太多无关特征属性时，极其容易导致决策树出现过拟合情况。基于上述情况，我们可以通过合理、有效地取样来抑制噪声的干扰，同时也可以主动删减一些特征属性，使得特征属性的数量保持一个合理的范围，从而避免过拟合。

5.1.2 决策树算法问题

在决策树算法的改进上，我们最主要的方法就是剪枝，删减掉多余的决策分支。剪枝的具体原理与思路见1.2.5。此外，我们可以设置决策树的最大深度，当某个决策分支达到最大深度时便不再决策，而是设立叶节点。

5.2 C4.5 相比 ID3 的优缺点

C4.5 决策树的提出是为了解决 ID3 决策树的一个缺点。当一个属性的可取值数目较多时，那么可能在这个属性对应的可取值下的样本只有一个或者是很少个，那么这个时候它的信息增益是非常高的，这个时候纯度很高，ID3 决策树会认为这个属性很适合划分，但是较多取值的属性来进行划分带来的问题是它的泛化能力比较弱，不能够对新样本进行有效的预测。而 C4.5 通过使用特征属性各取值的熵去除信息增益、得到信息增益率，从而很好地避免了 ID3 决策树的问题。

但是，C4.5 决策树也同样存在一些缺点，例如其更偏好可取值数目较少的特征属性；过于复杂的分支导致模型容易过拟合；多叉树的设置和计算每个节点时更大的计算量都导致了耗时长、模型效率不高等。

5.3 利用决策树进行特征选择

一般来说，当一个特征属性越早参与决策构造的节点分裂、出现次数越多时，其重要程度往往越高。此外，当一个特征属性在决策树中的平均信息增益/平均信息增益率越大，或者平均基尼指数越小时，特征的重要性也往往越高。因此，假设从根节点到任一个叶节点为一个路径，我们可以从以下几种方式衡量特征的重要性：

(1) 统计各个特征在所有路径上出现的次数，次数越多，重要程度越高（需要注意的是，一个节点可能因为处于不同路径上导致出现次数可以统计为多次。显然根节点的出现次数即为路径数）；

(2) 统计各个特征在所有路径上出现的先后位置，然后计算平均位置。如果特征在某条路径上没有出现的话，则默认其出现位置在最后。然后我们比较各个特征的平均出现位置，越靠前的话重要性越高；

(3) 统计各个节点的信息增益/信息增益率/基尼指数，然后计算平均值，表现越好的特征重要性越高；

在主流机器学习框架中，大多数都使用方式 (3) 来衡量特征的重要性，而 (1) 和 (2) 是本人自己瞎想的……除此以外，主流机器学习框架还会使用 OOB 数据等方式计算特征的重要性。由于本人尚未完全理解其中的概念和含义，故不再讲述。

6 实验总结与感想

本次实验是人工智能第 2 次实验，要求我们实现三种决策树，并对测试集进行预测。在实验过程中，我编写构造代码时遇到了很多在理论课上没有留意到的细节问题，给我的实验过程带来了一定困难，不过最后都顺利解决了。本次实验实现的算法准确率都有 90% 左右，一些决策树甚至能达到 95+%，相比上一次实验的准确率来说有了很大的提升。这也从某种程度上说明了决策树算法的经典性与优越性。

在此感谢老师和助教在实验过程中为我提供的帮助。希望我能顺利完成下一次实验。