人工智能实验报告（二）

- 院系/年级专业：数据科学与计算机学院/18级计科
- 学号/姓名：18340236/朱煜

一、实验题目

决策树

二、实验原理

1、决策树

- 决策树是一种树形结构，每个非叶节点表示一个特征属性上的测试，每个分支代表特征上的取值，每个叶节点代表最后输出的标签。
- 在分类问题中，表示基于特征对实例进行分类的过程。学习时，利用训练数据，根据损失函数最小化的原则建立决策树模型；预测时，对新的数据，利用决策模型进行分类。
- 决策树学习的算法通常是一个递归地选择最优特征，并根据该特征对训练数据进行分割，使得各个子数据集有一个最好的分类的过程。这一过程对应着对特征空间的划分，也对应着决策树的构建，决策树构建有以下过程：
    1. 构建根节点，将所有训练数据都放在根节点，选择一个最优特征，按着这一特征将训练数据集分割成子集，使得各个子集有一个在当前条件下最好的分类。
    2. 如果这些子集已经能够被基本正确分类，那么构建叶节点，并将这些子集分到所对应的叶节点去。如果没有特征可以继续分类，则进行多数投票得到分类。
    3. 如果还有子集不能够被正确的分类，那么就对这些子集选择新的最优特征，继续对其进行分割，构建相应的节点，如果递归进行，直至所有训练数据子集被基本正确的分类，或者没有合适的特征为止。
    4. 每个子集都被分到叶节点上，即都有了明确的类，这样就生成了一颗决策树。

2、ID3算法

（1）信息熵（经验熵）

- 如果一件事有k种可能的结果，每种结果的概率为$P_i, i=1,\dots,k$，在数据集D中表示为第k类样本所占的比例，则这件事代表的数据集D的信息熵为

$$H(D)=-\sum_{i=1}^{k}P_i log(P_i)$$

  并且规定$0log(0)=0$。
- 信息熵越大，表示样本越混乱；信息熵越小，表示样本越纯净。

（2）条件熵

- 条件熵表示在A发生的情况下，得到的D的熵，可表示在A发生后D的不确定性，计算公式为

$$H(D|A)=\sum_{a}P_a H(D|A=a)$$

- 条件熵是指在A的条件下，另一个变量D熵对A的期望

(3) 信息增益

- 信息增益表示得知特征A的信息而使得D集合的信息不确定性减少的程度。它为集合D的经验熵减去特征A的条件熵。公式表示为

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

- 信息增益大表明信息增多，信息增多，则不确定性就越小，意味着用属性A进行划分所获得的数据集纯度提升越大，这就是ID3算法选择最优划分属性的依据。

(4) ID3算法实现

- ID3算法的核心是在决策树的每一个非叶子结点划分之前，先计算每一个特征所带来的信息增益，选择其中最大信息增益的特征来划分该节点，因为信息增益越大，区分样本的能力就越强，越具有代表性。
- 在构建决策树的过程中利用信息增益原则进行特征选择，递归的构建决策树。从根节点开始，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，根据这个特征的不同取值建立子节点；递归的对子节点继续进行选择，最后构建成决策树。
- ID3算法伪代码如下

```
ID3DecisionTree(train_set, attribute_list)。//采用ID3算法生成决策树
输入：训练样本train_set，由离散值特征表示；特征的集合attribute_list。
输出：一棵决策树

decision_tree_node = TreeNode():
if train_set.value is same:
    decison_tree_node.value = train_set.value
    return decision_tree_node(leaf)//样例属于同一类，则作为叶节点直接返回
if attribut_list is empty://已无特征再进行划分，
    decison_tree_node.value = train_set.VoteValue()
    return decison_tree_node(leaf)//多数投票得到标签，作为叶节点返回
for attribute in attribute_list:
    attribute.CalRelativeEntropy()//对于每个特征计算信息增益

best_attribute = MaxRelativeEntropy()//得到信息增益最高的特征
decision_tree_node.attribute = best_attribute//标志叶节点
attribute_list.del(best_attribute)//删除已选择的特征

child_node = {}
for value in best_attribute.value():
    sub_train_set = train_set.Divide(value)//根据特征的取值重新划分训练样本
    child_node += ID3DecisionTree(sub_train_set, attribute_set)//递归生成子树
decision_tree_node.child_node=child_node//得到子节点
return decision_tree_node
```

3、C4.5算法

(1) 信息增益率

- 信息增益率是在信息增益的基础之上乘上一个惩罚参数。特征个数较多时，惩罚参数较小；特征个数较少时，惩罚参数较大。
- 惩罚参数为数据集D以特征A作为随机变量的熵的倒数。信息增益率$g_R(D,A)$具体计算公式为

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

其中$H_A(D)$为数据集D以特折A为随机变量得到的信息熵，由信息熵的计算公式得

$$H_A(D) = -\sum_a \frac{|D_a|}{|D|} \times log(\frac{|D_a|}{|D|})$$

（2）C4.5算法实现

- C4.5用**信息增益率**来选择特征，克服了用信息增益选择特征时偏向选择取值多的属性的不足，其建树过程与ID3算法相同。
- C4.5算法伪代码如下

```
C4p5DecisionTree(train_set, attribute_list)。//采用C4.5算法生成决策树
输入：训练样本train_set，由离散值特征表示；特征的集合attribute_list。
输出：一棵决策树

decision_tree_node = TreeNode():
if train_set.value is same:
    decison_tree_node.value = train_set.value
    return decision_tree_node(leaf)//样例属于同一类，则作为叶节点直接返回
if attribut_list is empty://已无特征再进行划分，
    decison_tree_node.value = train_set.VoteValue()
    return decison_tree_node(leaf)//多数投票得到标签，作为叶节点返回
for attribute in attribute_list:
    attribute.CalRelativeEntropy()//对于每个特征计算信息增益
    attribute.CalEntropy()//计算特征本身的信息熵

best_attribute = MaxgRation(relative_entropy, entropy)//得到信息增益比最高的特征
decision_tree_node.attribute = best_attribute//标志叶节点
attribute_list.del(best_attribute)//删除已选择的特征

child_node = {}
for value in best_attribute.value():
    sub_train_set = train_set.Divide(value)//根据特征的取值重新划分训练样本
    child_node += C4p5DecisionTree(sub_train_set, attribute_set)//递归生成子树
decision_tree_node.child_node=child_node//得到子节点
return decision_tree_node
```

4、CART算法

（1）GINI系数

- GINI指数为1与每一类别的概率平方之和的差值，反映了样本集合的不确定性程度，即 $基尼指数 = 样本被选中的概率 * 样本被分错的概率$。GINI指数越大，样本集合的不确定性程度越高。计算公式为

$$Gini(p) = \sum_{k=1}^{K} p_k(1-p_k) = 1 - \sum_{k=1}^{K} p_k^2$$

- 特征A的条件下，数据集D的GINI系数为

$$Gini(D, A) = \sum_a p(a) \times Gini(D_a|A=a)$$

其中

$$Gini(D_a|A=a) = \sum_{i=1}^{n} p_i(1-p_i) = 1 - \sum_{i=1}^{n} p_i^2$$

（2）CART算法实现

- CART算法使用**GINI系数**来选择特征，对于一个具有多个取值（超过2个）的特征，需要计算以每一个取值作为划分点，对数据集D划分之后子集的纯度$Gini(D, A_i)$，其中A_i 表示特征A的可能取值，然后从所有的可能划分的$Gini(D, A_i)$中找出GINI系数最小的划分，这个划分的划分点，便是使用特征A对样本集合D进行划分的最佳划分点。
- 用最佳划分点划分特征值取值后，CART建树则采用二分的方式，因此得到的树为二叉树。
- CART算法伪代码如下

```
CARTDecisionTree(train_set, attribute_list)。//采用CART算法生成决策树
输入：训练样本train_set，由离散值特征表示；特征的集合attribute_list。
输出：一棵决策树

decision_tree_node = TreeNode():
if train_set.value is same:
    decison_tree_node.value = train_set.value
    return decision_tree_node(leaf)//样例属于同一类，则作为叶节点直接返回
if attribut_list is empty://已无特征再进行划分，
    decison_tree_node.value = train_set.VoteValue()
    return decison_tree_node(leaf)//多数投票得到标签，作为叶节点返回
for attribute in attribute_list:
    //将特征取值进行二分，计算对应的GINI系数
    attribute_value_combinations = attribute_value_set.AllCombination()
    for combination in attribute_value_combinations:
        attribute.CalRtion(combination)//对于每个划分计算

best_attribute = MingRation(relative_entropy, entropy)//得到基尼系数最低的特征
//得到对应的最好特征划分
best_combination_left, best_combination_right =
GetCombination(best_attribute)
decision_tree_node.attribute = best_attribute//标志叶节点
attribute_list.del(best_attribute)//删除已选择的特征

child_node = {}
//根据特征的取值与特征取值组合划分训练样本
sub_train_set_left = train_set.Divide(best_combination_left, value)
sub_train_set_right = train_set.Divide(best_combination_right, value)
//根据特征的取值与特征取值组合划分训练样本
child_node += CARTDecisionTree(sub_train_set_left, attribute_set)//递归生成子
树
child_node += CARTDecisionTree(sub_train_set_right, attribute_set)//递归生成子
树
decision_tree_node.child_node=child_node//得到子节点
return decision_tree_node
```

三、实验过程

1、ID3算法

（1）关键代码

- 由于本次实验只有一个数据集，需要自行划分验证集与测试集。本次采用的为分层随机抽样，保证验证集中正反例的比例与数据集的比例相同，再通过调整验证集占总数据集的比例，以寻找正确率最高的占比，具体划分函数如下，而具体的数据集比例结果在实验分析处。

```
def Divide(data_set, proportion):
    # 计算原数据集中正反例的个数
    label_dict = {'0': 0, '1': 0}
    for i in range(1, len(data_set)):
        if data_set[i][-1] == '0':
            label_dict['0'] += 1
        else:
            label_dict['1'] += 1
    # 根据比例得到验证集中正反例应该有的数目
    for value in label_dict:
        label_dict[value] = int(label_dict[value]*proportion)
    select_set = [data_set[i] for i in range(1, len(data_set))]

    train_set = []
    test_set = []
    while len(select_set) > 0:
        # 生成随机数，随机抽样
        i = random.randint(0, len(select_set)-1)
        # 进行分层抽样
        if select_set[i][-1] == '0':
            if label_dict['0'] > 0:
                train_set.append(select_set[i])
                label_dict['0'] -= 1
                select_set.pop(i)
            else:
                test_set.append(select_set[i])
                select_set.pop(i)
        else:
            if label_dict['1'] > 0:
                train_set.append(select_set[i])
                label_dict['1'] -= 1
                select_set.pop(i)
            else:
                test_set.append(select_set[i])
                select_set.pop(i)
    return train_set, test_set
```

- ID3决策树采用多叉树的方式保存，具体节点结构如下。

```
class DecisionTreeNode:
    def __init__(self, leaf=False, value=None, attribute=None,
child_node=None):
        self.leaf = leaf                #是否为叶节点
        self.value = value              #叶节点则为最终取值
        self.attribute = attribute      #中间节点则为划分特征
        self.child_node = child_node    #子节点
```

- 为了方便计算不同标签的信息熵，信息熵计算函数的参数`attribute_index`确定计算哪一列的信息熵，默认为最后一行，则为标签的信息熵，实现代码如下，

```
def CalEntropy(data_set, attribute_index=-1):
    # 计算下标对应的特征的熵，默认计算标签
    attribute_dict = dict()
    # 得到每个特征取值的数量
    for i in range(len(data_set)):
        attribute = data_set[i][attribute_index]
        if attribute in attribute_dict:
            attribute_dict[attribute] += 1
        else:
            attribute_dict[attribute] = 1
    # 信息熵
    Entropy = 0.0
    for attribute in attribute_dict:
        # 计算熵并求和
        Entropy += (- attribute_dict[attribute]/len(data_set)
                    * math.log2(attribute_dict[attribute]/len(data_set)))
    return Entropy
```

在计算信息增益时，可根据特征划分子数据集传入以上函数，得到该特征取值的信息熵，再求和得到条件熵，根据公式可得到特征对应的信息增益，实现代码如下。

```
def CalRelativeEntropy(data_set, attribute_set, attribute_map):
    relative_entropy = []
    # 计算经验熵
    data_entropy = CalEntropy(data_set)
    for i in range(len(attribute_set)):
        attribute_value_set = attribute_map[attribute_set[i]]
        attribute_entropy = 0.0
        sub_data_set = []
        for value in attribute_value_set:
            sub_data_set = SubDataSet(data_set, i, value)
            # 计算每个特征下的条件熵
            attribute_entropy += (len(sub_data_set) /
                                  len(data_set))*CalEntropy(sub_data_set)
        # 计算信息增益
        relative_entropy.append(data_entropy - attribute_entropy)
    # 返回信息增益列表，与各特征对应
    return relative_entropy
```

- 根据以上函数可计算各标签的信息增益，在划分时便可得到最优的决策特征，再根据算法递归生成决策树，具体代码如下。

```
def DecisionTree(data_set, attribute_set, fuction, attribute_map):
    # 所有样本的标签属于同个一类型，则不需要再划分，作为叶节点放回
    if LabelIsSame([line[-1] for line in data_set]):
        return DecisionTreeNode(True, data_set[0][-1])
    # 若所有特征都已经用于划分，则进行多数投票
    if len(attribute_set) == 0:
        return DecisionTreeNode(True, VoteLabel([line[-1] for line in
data_set]))
    #CART算法
    if fuction == 'CART':
        #最好的特征作为决策节点，以及相应特征值划分
        best_attribute, merge = BestAttributeAndMerge(
            data_set, attribute_set, attribute_map)
        #根据特征值划分新的数据集与特征取值[value, 'other']
```

```
        data_set, attribute_value_set = MergeDataSet(
            data_set, merge, best_attribute)
        #新的特征集合
        new_attribute_set = [i for i in attribute_set]
        attribute_save = new_attribute_set.pop(best_attribute)
    #ID3算法和C4.5算法
    else:
        #最好的特征作为决策节点，以及相应特征值划分
        best_attribute = BestAttribute(
            data_set, attribute_set, fuction, attribute_map)
        #新的特征集合
        new_attribute_set = [i for i in attribute_set]
        attribute_save = new_attribute_set.pop(best_attribute)
        #根据特征值划分新的数据集与特征取值
        attribute_value_set = attribute_map[attribute_save]
    #根据特征取值获得子节点
    child_node = {}
    for value in attribute_value_set:
        #划分子数据集
        sub_data_set = SubDataSet(data_set, best_attribute, value)
        #子节点没有数据集则直接采用父节点的多数标签投票
        if len(sub_data_set) == 0:
            child_node[value] = DecisionTreeNode(
                True, VoteLabel([line[-1] for line in data_set]))
        else:
        #递归建树
            child_node[value] = DecisionTree(
                sub_data_set, new_attribute_set, fuction, attribute_map)
    #父节点指向子节点返回
    return DecisionTreeNode(attribute=attribute_save, child_node=child_node)
```

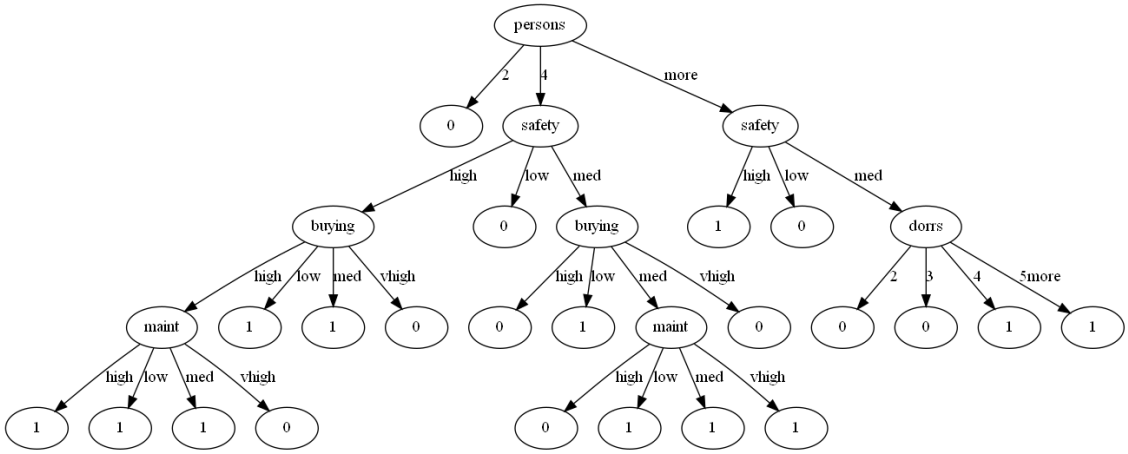
- 生成决策树之后，根据生成的决策树对测试集进行分类，具体代码如下，再根据结果与实际结果比较得到正确率

```
def Classify(test_set, decision_tree, fuction, attribute_set):
    result = []
    # CART算法
    if fuction == 'CART':
        for line in test_set:
            # 从根节点进入树
            line_node = decision_tree
            while line_node.leaf == False:
                test = True
                index = attribute_set.index(line_node.attribute)
                # 根据特征取值进行判断进入子节点
                for key in line_node.child_node:
                    if line[index] in key.split():
                        line_node = line_node.child_node[key]
                        test = False
                if test == True:
                    line_node = line_node.child_node['other']
            # 到达叶节点则生成结果
            result.append(line_node.value)
    # ID3和C4.5
    else:
        for line in test_set:
            # 从根节点进入
```

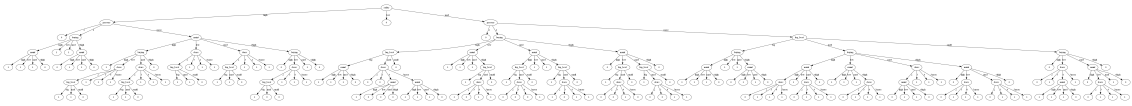
```
            line_node = decision_tree
            while line_node.leaf != True:
                index = attribute_set.index(line_node.attribute)
                # 根据特征取值进入子节点
                for key in line_node.child_node:
                    if line[index] == key:
                        line_node = line_node.child_node[key]

            result.append(line_node.value)
    return result
```

(2) 实验结果

- 为了方便查看生成的决策树，使用了`graphvizv`可视化图形工具输出决策树图像，具体函数实现在函数`PrintSubTree`与`PrintDecisionTree`中，为展示效果，当验证集占比为0.05时生成的ID3决策树如下，



当验证集占比为0.7时，生成的决策树如下，可见生成的决策树非常庞大，详细图片可见`ID3DesicionTree.png`文件



- 通过简单的分类函数，使用生成的ID3决策树进行分类，再与真实结果进行比较，当验证集占比为0.7时，得到的决策树模型正确率结果如下，可知ID3对该模型的拟合效果较佳

```
ID3正确率:  0.9672447013487476
```

2、C4.5算法

(1) 关键代码

- 在C4.5中，验证集与测试集的划分方式与ID3相同，由于C4.5与ID3同为多叉树，因此可以使用相同的树节点结构，实现代码同上。
- 根据信息增益率的计算公式，可通过计算每个特征的信息增益与自身信息熵的比值得到信息增益率，使用上述的`CalEntropy`与`CalRelativeEntropy`函数即可计算得到结果，实现代码如下，

```
def CalgRation(data_set, attribute_set):
    gRation = []
    # 计算每个特征的信息熵
    for i in range(len(attribute_set)):
        gRation.append(CalEntropy(data_set, i))
    # 计算每个特征对应的信息增益
    relative_entropy = CalRelativeEntropy(
        data_set, attribute_set, attribute_map)

    for i in range(len(gRation)):
        gRation[i] = relative_entropy[i] / (gRation[i] + 0.000000001)
    return gRation
```

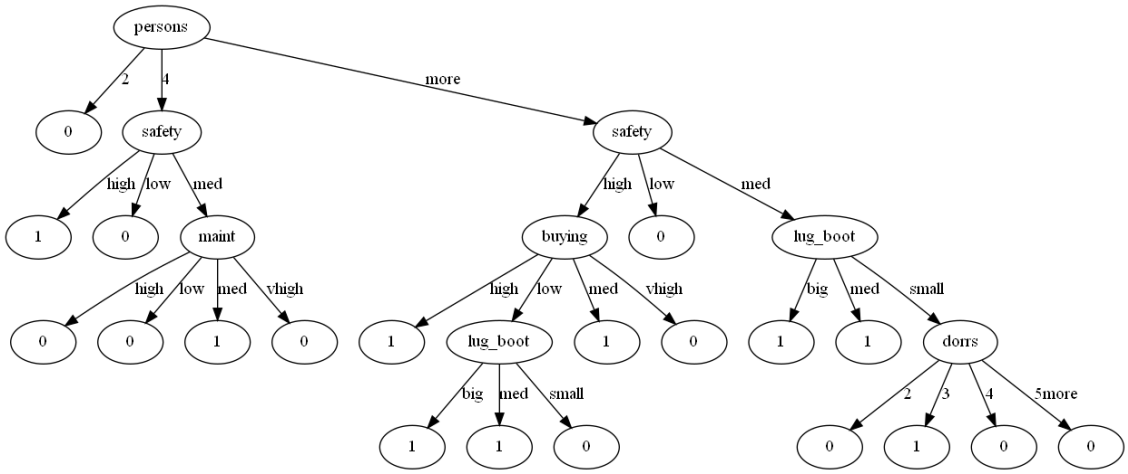
与ID3不同，C4.5根据计算得到的信息增益率选择最优标签，选择的代码较为简单，如下。

```
def BestAttribute(data_set, attribute_set, fuction, attribute_map):
    if fuction == 'ID3':
        relative_entropy = CalRelativeEntropy(
            data_set, attribute_set, attribute_map)
        return relative_entropy.index(max(relative_entropy))

    elif fuction == 'C4.5':
        gRation = CalgRation(data_set, attribute_set)
        return gRation.index(max(gRation))
```

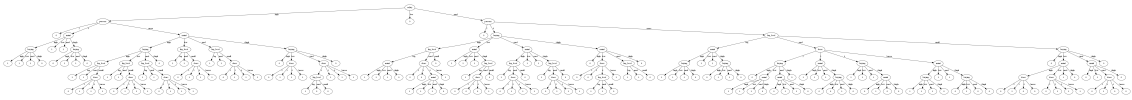
- 由于ID3与C4.5算法实现的区别仅在于标签的选择，其建树过程完全相同，则C4.5建树实现代码同上。

(3) 实验结果

- 当验证集占比为0.05时生成的决策树如下，



当验证集占比为0.7时生成的决策树如下，同样的非常庞大，详细图片可见`C4.5DesicionTree.png`文件



- 通过简单的分类函数，使用生成的C4.5决策树进行分类，再与真实结果进行比较，当验证集占比为0.7时，得到的决策树模型正确率结果如下，可知C4.5对该模型的拟合效果较佳，与ID3的正确率基本相同

```
C4.5正确率:  0.9653179190751445
```

3、CART算法

（1）关键代码

- 为了生成所有划分，根据传入的特征取值集合，生成全排列，其中删除全空与其本身，用以作为GINI系数二分的依据，具体代码如下，

```
def Combinations(value_set):
    # 生成value_set的全排列，除去全空与其本身
    combinations = [[]]
    for value in value_set:
        combinations += [subset + [value] for subset in combinations]
    return combinations[1:len(combinations)-1]
```

根据划分计算GINI系数，得到最好的特征与相对应的划分结果，GINI系数计算代码如下，

```
def CalGiniByVal(data_set, attribute_index, value_sub_set):
    # 根据划分得到四种样例的数目：划分内正例，划分内反例，划分外正例，划分外反例
    gini_p = [[0, 0], [0, 0]]
    for i in range(len(data_set)):
        if data_set[i][attribute_index] in value_sub_set:
            if data_set[i][-1] == '0':
                gini_p[0][0] += 1
            else:
                gini_p[0][1] += 1
        else:
            if data_set[i][0] == '0':
                gini_p[1][0] += 1
            else:
                gini_p[1][1] += 1
    # print(gini_p)
    # 根据公式计算划分得到的GINI系数
    gini_rela = [1-(gini_p[0][0]/(gini_p[0][0]+gini_p[0][1]+0.000000001)) **
                 2-(gini_p[0][1]/(gini_p[0][0]+gini_p[0]
[1]+0.000000001))**2]
    gini_rela += [1-(gini_p[1][0]/(gini_p[1][0]+gini_p[1][1]+0.000000001))
                  ** 2-(gini_p[1][1]/(gini_p[1][0]+gini_p[1]
[1]+0.000000001))**2]
    # 计算在特征A情况下，数据集的GINI系数
    gini_rela[0] *= sum(gini_p[0])/(sum(gini_p[0])+sum(gini_p[1]))
    gini_rela[1] *= sum(gini_p[1])/(sum(gini_p[0])+sum(gini_p[1]))
    return sum(gini_rela)
```

与ID3算法和C4.5算法不同，CART算法生成的树是二叉树，为了兼容树结构，需要合并特征取值，选择采用`value1 value2 value3`的方式，在每一需要合并特征取值直接加入空格得到新字符串，其余特征取值采用`other`代替，这样得到的特征取值只有两种，能有效地兼容树的结构，实现代码如下

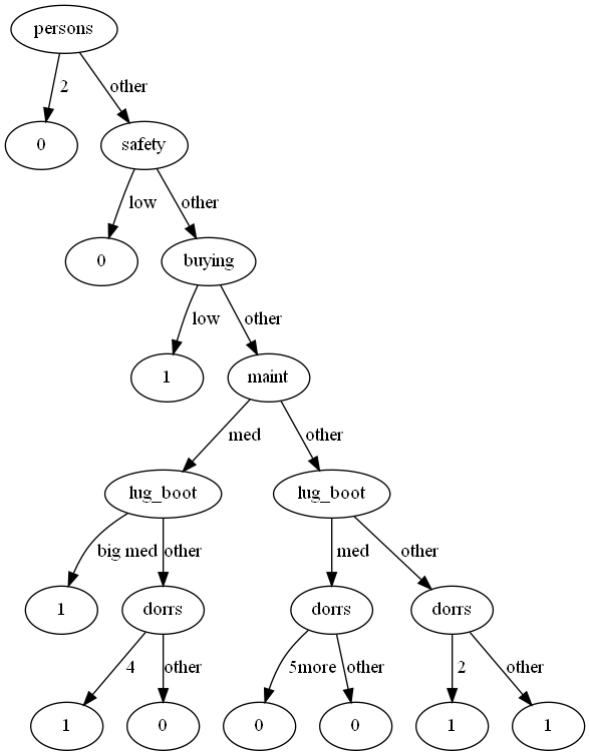
```
def MergeDataSet(data_set, merge, best_attribute):
    # 合并后的特征值字符串
    merge_str = str()
    # 除外的特征值字符串
    other_str = 'other'
    for i in merge:
        merge_str = merge_str + ' ' + i
```

```
    # print(merge_str)
    for line in data_set:
        if line[best_attribute] in merge:
            line[best_attribute] = merge_str
        else:
            line[best_attribute] = other_str
    return data_set, [merge_str, other_str]
```

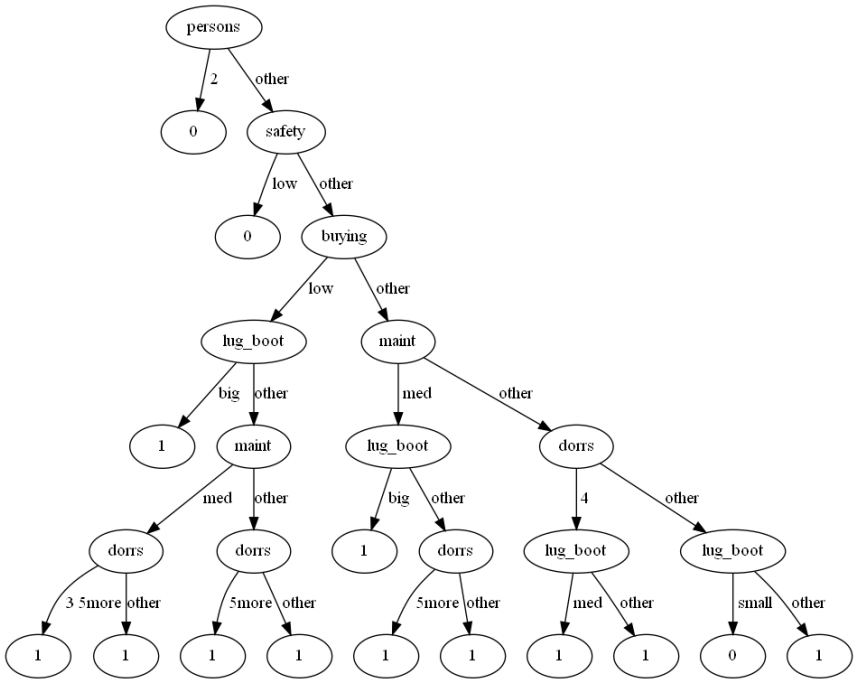
- 由于对数据集进行特殊的修改，CART算法的建树过程可使用与ID3、C4.5相同的函数，实现代码同上。

(3) 实验结果

- 当验证集占比为0.05时生成的决策树如下，



当验证集占比为0.7时生成的决策树如下，可知CART算法生成的决策树较ID3与C4.5结构简单



- 通过简单的分类函数，使用生成的CART决策树进行分类，再与真实结果进行比较，当验证集占比为0.7时，得到的决策树模型正确率结果如下，可知CART对该模型的拟合效果没有ID3与C4.5好，因为生成的树结构较为简单，而叶节点多采用多数投票的原则生成标签，因此效果并不好

```
CART正确率：  0.884393063583815
```

四、实验分析

1. 为了选取最好的验证集占比，选取了多个占比进行决策树的建立，并计算相应正确率，得到的正确率结果大致如下，可得到当验证集划分在0.7~0.9之间时，得到的准确率最高。

```
正确率如下
占比        ID3         C4.5        CART
0.100000  0.903599  0.887532  0.859254
0.200000  0.910340  0.947939  0.879971
0.300000  0.923140  0.912397  0.876860
0.400000  0.950820  0.919961  0.860174
0.500000  0.962963  0.949074  0.848380
0.600000  0.959538  0.955202  0.854046
0.700000  0.967245  0.959538  0.895954
0.800000  0.962536  0.965418  0.835735
0.900000  0.976879  0.971098  0.838150
```

```
正确率如下
占比        ID3         C4.5        CART
0.700000  0.963391  0.961464  0.863198
0.720000  0.954639  0.962887  0.849485
0.740000  0.968889  0.951111  0.886667
0.760000  0.937500  0.954327  0.865385
0.780000  0.963255  0.939633  0.874016
0.800000  0.959538  0.968208  0.864162
0.820000  0.977564  0.951923  0.884615
0.840000  0.956679  0.960289  0.844765
0.860000  0.971193  0.975309  0.868313
0.880000  0.918660  0.956938  0.856459
```

2. **创新点**：为了提高准确度，采用随机森林的模型，多次以0.7为占比随机选取训练集，生成ID3，C4.5，CART三种决策树，每次使用相同的测试集进行测试，保存每一步ID3，C4.5，CART树分类后的结果，对树生成的结果进行多数投票得到最终结果，以投票得到的结果作为森林最后的结果，最终得到的结果正确率如下，可得随机森林得到的分类结果基本完全正确

```
此时的p为:0.700000
ID3正确率：  0.987146529562982
C4.5正确率：  0.987146529562982
CART正确率：  0.8579691516709511
此时的p为:0.700000
ID3正确率：  0.9929305912596401
C4.5正确率：  0.9929305912596401
CART正确率：  0.890745501285347
此时的p为:0.700000
ID3正确率：  0.9865038560411311
C4.5正确率：  0.9890745501285347
CART正确率：  0.8798200514138818
此时的p为:0.700000
ID3正确率：  0.993573264781491
C4.5正确率：  0.993573264781491
CART正确率：  0.87146529562982
此时的p为:0.700000
ID3正确率：  0.9813624678663239
C4.5正确率：  0.9813624678663239
CART正确率：  0.8663239074550129
此时的p为:0.700000
ID3正确率：  0.9832904884318766
C4.5正确率：  0.9865038560411311
CART正确率：  0.8663239074550129
此时的p为:0.700000
ID3正确率：  0.980719794344473
C4.5正确率：  0.980719794344473
CART正确率：  0.8740359897172236
此时的p为:0.700000
ID3正确率：  0.9903598971722365
C4.5正确率：  0.993573264781491
CART正确率：  0.8817480719794345
最终随机森林得到的结果正确率：  0.993573264781491
```

五、实验思考

- 决策树有哪些避免过拟合的方法?
    1. 约束决策树：可通过设置每个节点的最小样本数，避免过拟合；设置树的最大深度。
    2. 有效地抽样：用相对能反映数据集特征的训练集去生成决策树。
    3. 预剪枝：在决策树生成过程中进行，对于后续节点，判断是否继续划分。
    4. 后剪枝：生成完整决策树后，自底向上地进行考察，假如某非叶节点变成叶节点时，决策树的准确率不降低，则变成叶节点。
- C4.5相比于ID3的优点是什么，C4.5又可能有什么缺点?
    - C4.5的优点：
        1. 通过信息增益率选择属性，克服了用信息增益时偏向选择取值多属性的问题。
        2. 处理连续型数据的属性，对连续属性离散化。
        3. 能够进行剪枝操作。
        4. 能够对空缺值进行处理。
    - C4.5的缺点：
        1. 对连续属性值需要扫描排序，会使C4.5性能下降。
        2. C4.5偏向于特征值较少的特征。
- 如何用决策树来进行特征选择（判断特征的重要性）
    1. 在随机森林中，可通过计算每个特征在随机森林中的每棵树做出的贡献之和再取平均值，得到的特征贡献结果进行比较，通常使用不纯度降低程度与袋外数据错误率来作为指标衡量。
    2. **不纯度**：在随机森林中对于每颗树，按照不纯度（GINI系数/信息增益率/信息熵）给特征排序，然后整个森林取平均。不纯度在分类中通常为GINI不纯度或信息增益/信息熵，对于回归问题来说是方差。 这样得到的特征排序结果则可以判断特征的重要性。
    3. **袋外数据错误率**：在随机森林中，直接测量每种特征对模型预测准确率的影响，基本思想是重新排列某一列特征值的顺序，观测降低了多少模型的准确率。对于不重要的特征，这种方法对模型准确率的影响很小，但是对于重要特征却会极大降低模型的准确率。具体步骤如下：
        - 对于随机森林中的每一颗决策树，使用相应的OOB数据（袋外数据）来计算它的袋外数据误差，记为$err_1(OBB)$。
        - 随机地对袋外数据所有样本的特征X加入噪声干扰，随机改变样本在特征X处的值，再次计算它的袋外数据误差，记为$err_2(OBB)$.
        - 假设随机森林中有N棵树，那么特征X的重要性$=\sum_n(err_2(OOB)-err_1(OOB))/N$，之所以可以用这个表达式来作为相应特征的重要性的度量值是因为：若给某个特征随机加入噪声之后，袋外数据的准确率大幅度降低，则说明这个特征对于样本的分类结果影响很大，也就是说它的重要程度比较高。