# 人工智能实验二

18340146 计算机科学与技术 宋渝杰

## 任务一：

实现 ID3，C4.5，CART 三种决策树

本次数据为 car_train.csv 每个文件有7列，前6列为特征（都为离散型），最后一列是标签 (0 or 1)

请自行分好训练验证集（在报告里说明怎么分的）， 评测指标为验证集上的准确率。

因此，本次实验的任务为：对数据 car_train.csv 分为训练集和验证集，对训练集生成三种决策树，并用验证集测试三种决策树的准确率

### 算法原理：

**数据集的分类**：选取前 t 条数据为训练集，其余 1728-t 条数据为验证集（ t 为可变参数）

本次任务算法分为三部分：建树、验证/测试、三种决策树特征选择

**建树部分**：建树主要分为五个小部分：

1. 初始化：创建根节点，它拥有所有特征和所有数据
2. 选择特征：遍历当前结点的特征和数据：
    - 如果当前所有数据标签相同，则该节点为叶节点，标签为所有数据相同的标签，结束当前节点
    - 如果标签不相同，则根据**某种原则（ID3、C4.5、CART）**，选择一个决策点
    - 如果根据上述原则无法选出决策点，则该节点为叶节点，根据多数原则确定标签，结束当前节点
3. 划分数据：根据选择出的决策点将当前数据集划分为两个或多个子数据集
4. 创建节点：为每个子数据集创建一个子节点，并分别连接根节点和每个子节点
5. 递归建树：对每一个子节点进行步骤二

**验证/测试部分**：该算法主要分为三个小部分

1. 初始化：从决策树的根节点开始比较
2. 比较：判断当前节点的状态：
    - 如果当前节点为叶节点，则进行下一步的验证/测试操作
    - 如果当前节点非叶节点，则根据当前节点的特征，寻找验证/测试数据该特征的取值，确定分支
        - 如果存在与验证/测试数据该特征的取值相同的分支，则更新当前节点为分支节点，继续进行比较操作
        - 如果不存在与验证/测试数据该特征的取值相同的分支，则根据当前节点的数据集的所有标签，根据多数原则确定标签，并进行验证/测试操作
3. 验证/测试：
    - 验证：对比验证数据的标签和叶节点的标签，相同的话即验证正确
    - 测试：确定该测试数据的标签为叶节点的标签

**三种决策树特征选择部分**：

- ID3：它的决策策略为**信息增益**，具体步骤为：
    1. 计算数据集 D 的**经验熵**：$H(D)=-\sum_{d\in D}p(d)\log_2 p(d)$，其中 $p(d)$ 为标签取值为 d 的数据量占总数据量的比例，这里（以及之后）的 log **以 2 为底**
    2. 计算特征 A 对数据集 D 的**条件熵**：$H(D|A)=\sum_{a\in A}p(a)H(D|A=a)$，其中 $p(a)$ 为特征 A 取值为 a 的数据量占总数据量的比例，$D|A=a$ 为数据集 D 特征 A 的取值为 a 的一个子集
    3. 计算**信息增益**：$g(D,A)=H(D)-H(D|A)$
    4. 选择**信息增益最大**的特征作为决策点
- C4.5：它的决策策略为**信息增益率**，具体步骤为：
    1. 计算特征 A 对数据集 D 的**信息增益**：$g(D,A)=H(D)-H(D|A)$
    2. 计算数据集 D 关于特征 A 的值的**熵**：$\mathrm{SplitInfo}(D,A)=-\sum_{j\in v}\frac{|D_j|}{|D|}\log_2\frac{|D_j|}{|D|}$，其中 $v$ 为特征 A 的各种取值，$|D_j|$ 为特征 A 取值为 $j$ 的数据量，$|D|$ 为总数据量
    3. 计算**信息增益率**：$\mathrm{gRatio}(D,A)=(H(D)-H(D|A))/\mathrm{SplitInfo}(D,A)$
    4. 选择**信息增益率最大**的特征作为决策点
- CART：它的决策策略为**GINI系数**，具体步骤为：
    1. 计算数据集 D 特征 A 取值为 a 的条件下的 GINI 系数：

$$\mathrm{gini}(D,A=a)=\frac{|D_{A=a}|}{|D|}\mathrm{gini}(\mathrm{D_{A=a}})+\frac{|D_{A!=a}|}{|D|}\mathrm{gini}(\mathrm{D_{A!=a}})$$
$$\mathrm{gini}(D)=2p(1-p)$$

       其中 $p$ 为参数数据集中标签为 1 的频率

    2. 选择**GINI系数最小**的特征及其取值作为决策点

## 伪代码/流程图：

训练：读取文件前 t 行数据（不包括第一行无效数据）-> 以该数据集构建三种树（构建过程已在上文说明）

验证：读取剩余 1728-t 行数据 -> 对每一行验证数据分别进行三种树的验证操作（验证过程已在上文说明）-> 搜集三种树的验证正确率并输出

测试：读取助教给的测试数据 -> 取前 15 行建 CART 树 -> 使用 CART 树对后 3 行测试数据进行测试操作（测试过程已在上文说明）-> 将测试结果以文件形式输出

## 代码展示：

下面仅展示关键代码，全部代码请移步 lab2.py 文件

读取训练数据，生成训练集：

```
def readTrain(f,t): # 读取训练数据
    feature = f.readline().strip('\n').split(',') # 读取特征
    data = []
    for i in range(t):
        data.append(f.readline().strip('\n').split(',')) # 读取数据
    return data,feature
```

划分子数据集：

```
def splitData(data,index,value): # 筛选特征index等于value的数据
    rData = []
    for l in data: # 遍历原数据所有行
        if l[index] == value:
            rData.append(l[:index]+l[index+1:]) # 同时删除特征index
    return rData
```

```
def splitData2(data,index,value): # 筛选特征index不等于value的数据
    rData = []
    for l in data: # 遍历原数据所有行
        if l[index] != value:
            rData.append(l[:index]+l[index+1:]) # 同时删除特征index
    return rData

def selectData(data,index,value): # 筛选特征index等于value的数据
    rData = []
    for l in data: # 遍历原数据所有行
        if l[index] == value:
            rData.append(l[:]) # 不删除特征index
    return rData

def selectData2(data,index,value): # 筛选特征index不等于value的数据
    rData = []
    for l in data: # 遍历原数据所有行
        if l[index] != value:
            rData.append(l[:]) # 不删除特征index
    return rData
```

计算信息熵:

```
def HD(data): # 信息熵 H(D)
    D = {}
    for l in data: # 统计0和1的个数
        if l[-1] not in D.keys():
            D[l[-1]] = 0
        D[l[-1]] += 1
    ans = 0.0
    for key in D:
        p = D[key]/len(data) # 按照公式算～
        ans -= p*log(p,2) # p(d)log(p(d))，这里以2为底
    return ans
```

计算条件熵:

```
def HD_A(data,index): # 条件熵 H(D|A)
    Vals = set([l[index] for l in data]) # 计算特征index有多少取值
    ans = 0.0
    for value in Vals: # 遍历a
        subData = splitData(data,index,value) # 划分出关于特征index的子数据集
        p = len(subData)/len(data) # p(a)
        ans += p*HD(subData) # p(a)*H(D|A=a)
    return ans
```

计算信息增益:

```
def GD_A(data,index): # 信息增益 g(D,A)
    return HD(data) - HD_A(data,index)
```

ID3:

```
def ID3(data): # ID3
    maxGD_A = 0.0
    bestIndex = -1 # 信息增益最大的特征下标
    for index in range(len(data[0])-1): # 遍历所有特征
        gd_a = GD_A(data,index) # 计算信息增益
        if gd_a > maxGD_A:
            maxGD_A = gd_a
            bestIndex = index
    return bestIndex # 返回决策点特征下标
```

C4.5:

```
def SplitInfo(data,index): # SplitInfo(D,A)
    Vals = set([l[index] for l in data]) # 计算特征index有多少取值
    ans = 0.0
    for value in Vals: # 遍历a
        subData = splitData(data,index,value) # 划分出关于特征index的子数据集
        p = len(subData)/len(data) # p(a)
        ans -= p*log(p,2) # p(a)*log(p(a))，这里以2为底
    return ans

def C45(data): # C4.5
    maxGD_A_Split = 0.0
    bestIndex = -1 # 信息增益率最大的特征下标
    for index in range(len(data[0])-1): # 遍历所有特征
        gd_a_Split = GD_A(data,index)/SplitInfo(data,index) # 计算信息增益率
        if gd_a_Split > maxGD_A_Split:
            maxGD_A_Split = gd_a_Split
            bestIndex = index
    return bestIndex # 返回决策点特征下标
```

CART:

```
def GINI(data,index):
    Vals = set([l[index] for l in data]) # 计算特征index有多少取值
    bestGINI = 1e6
    bestVal = -1
    for value in Vals: # 遍历a
        subData1 = splitData(data,index,value) # 划分出关于特征index=value的子数据集
        subData2 = splitData2(data,index,value) # 划分出关于特征index!=value的子数据
集
        p1 = len(subData1)/len(data) # p(A=a)
        p2 = len(subData2)/len(data) # p(A!=a)
        if p1 == 1: # 说明之前成为过最优分割点
            continue
        num1 = [l[-1] for l in subData1]
        num2 = [l[-1] for l in subData2]
        ans = p1*(2*num1.count('1')/len(subData1)*num1.count('0')/len(subData1))
#2p(1-p)
        ans += p2*
(2*num2.count('1')/len(subData2)*num2.count('0')/len(subData2))
        if ans < bestGINI:
            bestGINI = ans
            bestVal = value
    return bestGINI,bestVal
```

```
def CART(data): # CART
    minCART = 1e6
    bestIndex = -1 # 基尼指数最小的特征下标
    bestValue = -1
    for index in range(len(data[0])-1): # 遍历所有特征
        cart,value = GINI(data,index)
        if cart < minCART:
            minCART = cart
            bestIndex = index
            bestValue = value
    return bestIndex,bestValue # 返回决策点特征下标和对应取值
```

建树：

```
def builtTree(data,feature,Func): # ID3、C4.5
    node = [l[-1] for l in data]
    if node.count(node[0]) == len(node): # 全为一种标签
        return node[0]
    if len(data[0]) == 1: # 只剩一种特征
        return cnt(node)
    bestIndex = Func(data) # 选出决策点
    bestFeature = feature[bestIndex] # 决策点特征
    tree = {bestFeature:{}}
    Vals = set([l[bestIndex] for l in data]) # 这个特征的所有取值
    for value in Vals:
        copyFeature = feature[:bestIndex]+feature[bestIndex+1:] # 去除这个特征
        tree[bestFeature][value] =
builtTree(splitData(data,bestIndex,value),copyFeature,Func) # 递归建树
    return tree

def builtTreeUsingCART(data,feature): # CART
    node = [l[-1] for l in data]
    if node.count(node[0]) == len(node): # 全为一种标签
        return node[0]
    bestIndex,bestVal = CART(data) # 选出决策点，最优切分点
    if bestIndex == -1: # 无法继续切分
        return cnt(node) # 多数原则判断标签
    bestFeature = feature[bestIndex] # 决策点特征
    tree = {bestFeature:{}}
    data1 = selectData(data,bestIndex,bestVal) # 划分子数据集（A=a）
    tree[bestFeature][bestVal] = builtTreeUsingCART(data1,feature) # 递归建树
    data2 = selectData2(data,bestIndex,bestVal) # 划分子数据集（A!=a）
    tree[bestFeature]['not '+bestVal] = builtTreeUsingCART(data2,feature) # 递归建
树
    return tree
```

验证过程：

```
def validation(vali,tree,m,data): # 验证
    num = 0 # 正确个数
    for row in vali: # 遍历验证集所有行
        copyTree = copy.deepcopy(tree)
        copyData = copy.deepcopy(data)
        while 1:
            j = 0 # 判断是否有路可走
            k = list(copyTree.keys())[0]
```

```
            for k2 in copyTree[k].keys(): # 遍历目前节点所有分支
                #print(k,vali[m[k]],k2) # 当前节点、要走的分支、当前遍历的分支
                if row[m[k]] == k2 or (k2.split(' ')[0] == 'not' and row[m[k]]
!= k2.split(' ')[1]): # 验证特征与该分支符合
                    copyTree = copyTree[k][k2] # 走到下一个节点
                    copyData = selectData(copyData,m[k],k2) # 筛选出满足走过的路所有
条件的数据集
                    j = 1
                    break
            if j == 0: # 无路可走
                node = [l[-1] for l in copyData] # 统计标签
                if row[-1] == cnt(node): # 多数原则判断后和验证标签相同
                    num += 1
                break
            if type(copyTree) == str: # 走到叶节点
                if copyTree == row[-1]:
                    num += 1
                break
    return num/len(vali) # 返回正确率
```

## 实验结果以及分析：

**调参过程**：为了调整训练集大小 t 以达到最优验证率，现对参数 t 进行调参：

| 训练集 | 验证集 | 训练 : 验证 | ID3 | C4.5 | CART |
|---|---|---|---|---|---|
| 346 | 1382 | 20% : 80% | 92.98% | 90.30% | 95.95% |
| 692 | 1036 | 40% : 60% | 93.34% | 93.53% | 97.10% |
| 864 | 864 | 50% : 50% | 93.17% | 93.87% | 97.22% |
| 1037 | 691 | 60% : 40% | 94.79% | 94.79% | 98.84% |
| **1210** | **518** | **70% : 30%** | **95.75%** | **95.75%** | **99.03%** |
| 1382 | 346 | 80% : 20% | 95.09% | 95.09% | 98.55% |
| 1555 | 173 | 90% : 10% | 96.53% | 95.38% | 97.69% |
| 1642 | 86 | 95% : 5% | 98.84% | 96.51% | 98.84% |
| **1711** | **17** | **99% : 1%** | **100.00%** | **100.00%** | **100.00%** |

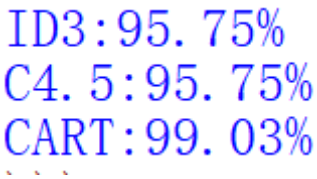
从这个表格可以看出，随着训练集比例的扩大，验证率先递增，之后有一小段稍微递减，最后递增至 100%。其中一个**极大点为 t = 1210**，此时训练 : 验证为 **70% : 30%**。

查阅相关资料得知（如西瓜书《机器学习》）：**当数据量比较小时（不过万），一般采用 7:3 的比例划分训练集和验证集。**而这次的实验结果也和该理念相符。

而对于后续训练集过大，验证集过小时，虽然表面上验证率增加，但是由于分母过小，验证不能大范围的覆盖模型，难以搜索出模型的所有问题，导致验证结果的可信度不会太高。

因此个人认为"评测指标为验证集上的准确率"存在一些不合理性（我可以极端的声明验证集大小为 1，且恰好验证正确，即直接 100% 验证准确率）

最后个人确定参数 t（验证集大小）为 **1210**，即全部数据的 **70%**，最终三种树的验证准确率为 **95.75%、95.75%、99.03%**




同时，在调参过程中，CART 树的准确率均高于或等于另外两种树，符合其二叉树判断更精准的性质

## 思考题：

**决策树有哪些避免过拟合的方法?**

- 剪枝：提前停止树的增长或者对已经生成的树按照一定的规则进行后剪枝。
- 约束决策树：设置每个叶子节点的最小样本数、设置树的最大深度、设置评估分割数据是的最大特征数量等等

**C4.5相比于ID3的优点是什么，C4.5又可能有什么缺点?**

- 优点：可以校正 ID3 偏向选择子类别多的特征的问题；可以处理连续变量
- 缺点：时间耗费大，算法低效；只适合于能够驻留于内存的数据集，当训练集大得无法在内存容纳时程序无法运行。

**如何用决策树来进行特征选择(判断特征的重要性)?**

- ID3 通过信息增益判断特征的重要性（信息增益越大，特征越重要）
- C4.5 通过信息增益率判断特征的重要性（信息增益率越大，特征越重要）
- CART 通过GINI系数判断特征的重要性（GINI系数越小，特征越重要）