# 机器学习第一次作业

# 《统计学习》+《机器学习》

# 实验报告



专业： 智能科学与技术

班级： 2017211316

学号： 2017211576

姓名： 董亚飞

2019年10月10日

## 实验目的

1. 通过编程与代码实现对决策树的生成过程有更直观与深入的了解
2. 在功能实现的过程中，对决策树模型训练过程的“数据处理—>拟合（含剪枝）—>预测”的有更系统性的掌握
3. 通过实验，对信息熵、信息增益（比）、基尼指数、均方误差等决策树相关评价标准以及相应算法的实现有更深入的掌握
4. 锻炼使用python语言实现机器学习相关算法的能力以及掌握Numpy和Pandas相关python库在数据科学分析与处理过程中的作用

## 实验环境与使用的Python库

Python 3.7 + Anaconda 3 + Visual Studio Code / PyCharm Community 2019.2

Python库：

numpy + pandas + math + sklearn + colections

## 实验内容

（点击跳转）

《统计学习》 [5.1](#_T 5.1) + [5.2](#_T 5.2)

《机器学习》（西瓜书） [4.3](#_T4-3) + [4.4](#_T4-4) + [4.6](#_T4-6)

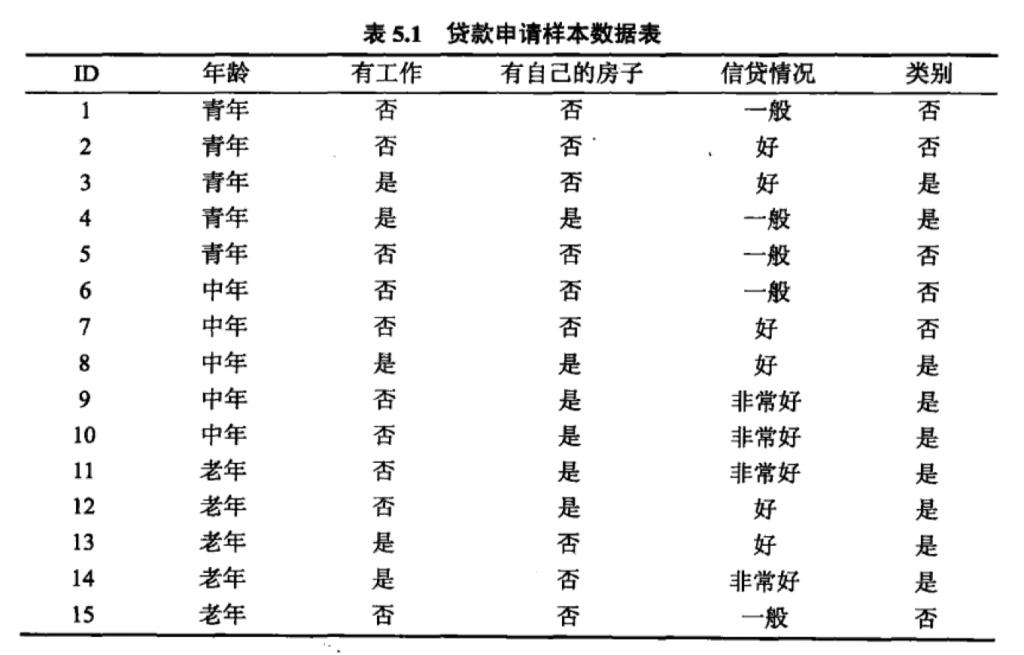
## 具体实现

### 《统计学习》部分

#### T 5.1

问题描述：

根据表5.1所给的训练集数据，利用信息增益比（C4.5）算法生成决策树



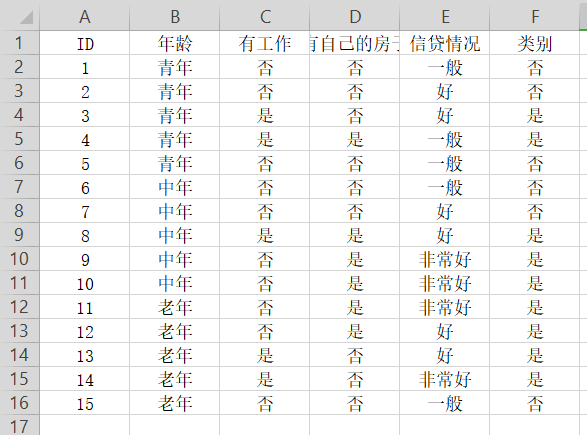
问题解决过程：

首先，题目要求使用C4.5算法，以信息增益比作为评估方法来生成决策树。决定选用python语言编程实现

代码实现过程

1. 数据预处理

为了将训练数据与代码分离以便于之后进行进一步的处理，我将表中对应的的数据在Excel中编辑并保存为csv格式文件便于python读取与处理



1. 处理思路（模块讲解及算法流程图展示）
2. 模块划分

**Data\_preprocess** : 数据预处理模块，从csv文件中读取数据并获取有用的X, y等相关信息

**Gain\_ratio**: 信息增益比计算模块，对根据某一属性划分后的数据集求出其信息增益比作为比较依据

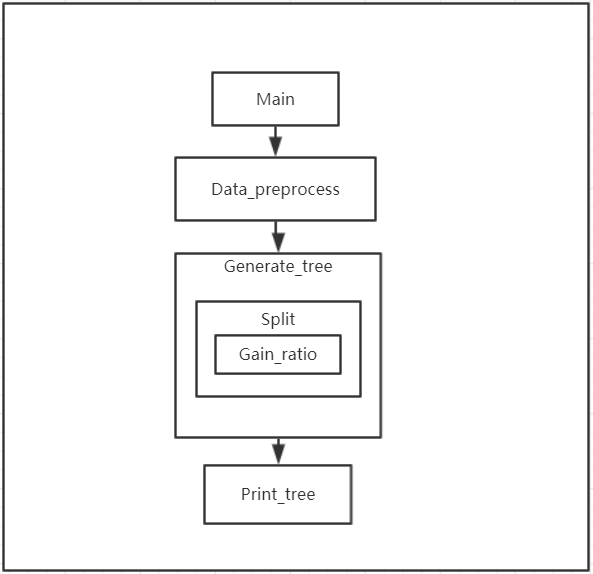
**Split**: 数据划分模块，在数据集上根据现有的可划分属性集合来对数据进行最优标准划分

**Generate\_tree:** 生成决策树模块，调用其他模块，输入初始训练数据集与初始可用划分属性集，生成训练模型决策树

**Print\_tree:** 根据所得的树结构，采用目录表示法输出所得决策树

**Main:** 主模块，调用其余模块完成训练任务

1. 算法模块关系图



1. 算法重点实现部分

* 使用的python库

*import* numpy *as* np #数据存储与处理

*import* pandas *as* pd #csv文件的读取与numpy.array的对接

*from* collections *import* Counter #信息熵计算及样本类别提取

*from* math *import* log #信息熵计算

* 决策树的数据结构

class DecisionTree:

    def \_\_init\_\_(*self*, *kind*, *dim*=-1):

        """决策树类，属性为决策结果（kind）和切分维度（dim）"""

        self.kind = kind

        self.dim = dim

        self.children = {}

将决策树定义为类，既可表示分支节点，也可表示叶节点。

kind: 值为数据的类别（y的取值，代表该节点表示的类别）

dim: 值为（int）,表示该节点划分数据集依据的属性维度，其中当dim为-1时表示该节点为叶节点，其余值为分支节点

children: 值为字典，储存在该属性维度（dim）下数据的属性取值（key）,及其对应的子树（value）

* 信息增益比计算

def entropy(*y*):

    """计算信息熵"""

    counter = Counter(y)

    res = 0

*for* num in counter.values():

        p = num / len(y)

        res += -p \* log(p)

*return* res

def gain\_ratio(*y*, *y\_s*):

    """对于针对某个特征进行划分后的数据集求信息增益比"""

    condition\_entropy = 0

*for* yc in y\_s:

*if* yc.size!=0:

            p = yc.size / len(y)

            condition\_entropy += p \* entropy(yc)

*return* 1 - condition\_entropy / entropy(y)

使用entropy(y)函数求出y的信息熵，gain\_ratio函数通过对ebtropy()函数的调用求出相应的条件熵，进而求出数据集y在划分为y\_s(y的列表)之后的信息增益比

* 数据集划分

def split(*X*, *y*, *d*):

    """根据在d维的取值不同进行划分"""

    X\_s, y\_s = [], []

*for* k in attributions[d]:

        X\_s.append(X[X[:, d] == k])

        y\_s.append(y[X[:, d] == k])

*return* X\_s, y\_s

def try\_split(*X*, *y*, *D*):

    """尝试进行划分，需要返回最佳划分维度d"""

    best\_gain\_r = float('-inf')

    best\_d = -1

*for* d in D:

        X\_s, y\_s = split(X, y, d)

        gain\_r = gain\_ratio(y, y\_s)

*if* gain\_r > best\_gain\_r:

            best\_gain\_r, best\_d = gain\_r, d

*return* best\_gain\_r, best\_d

首先设定一个函数用于求解数据集X,y在d维的不同取值划分，最后通过启发式算法遍历以信息增益比为依据寻找最优划分维度

* 树的生成

def generate\_DT(*X*, *y*, *D*):

    """传入输入数据，输出数据与可用属性集合，递归进行决策树生成"""

    kind = Counter(y).most\_common(1)[0][0]

*#样本属于同一类别*

*if* entropy(y) == 0:

*return* DecisionTree(kind)

*#属性集已空*

*if* len(D) == 0:

*return* DecisionTree(kind)

*#所有样本在已知属性集上取值相同*

*else*:

        flag = True

*for* d in D:

*if* len(Counter(X[:,d])) != 1:

                flag = False

*break*

*if* flag:

*return* DecisionTree(kind)

*else*:

*#正常进行划分*

            gain\_r, d = try\_split(X, y, D)

            X\_s, y\_s = split(X, y, d)

            D\_ = D.copy()

            D\_.remove(d)

            DT = DecisionTree(kind, d)

*for* i in range(len(X\_s)):

*#遇到空样本集*

*if* X\_s[i].size == 0:

                    t = DecisionTree(kind)

*else*:

                    t = generate\_DT(X\_s[i], y\_s[i], D\_)

                DT.children[attributions[d][i]] = t

*return* DT

通过递归调用实现决策树的生成，首先考虑递归的终止条件，及相应的叶节点生成

—— 样本属于同一类别（kind为样本最多类别）

—— 可用属性集已经为空（kind为样本最多类别）

—— 剩余样本在所给属性集上的取值都相同（kind为样本最多类别）

如果未达到叶节点，则可进行“分支”，对该维度的每个属性值都创建一个分支（调用划分模块函数），并创建相应子树，对某一属性：

没有样本集，生成子树（kind为父节点的kind <先验概率分布>）

有样本集，递归调用创建子树（此时属性集中需要剔去使用过的属性）

* 树的输出

def print\_tree(*T*, *layer*=0):

    """合理输出决策树"""

*if* T.dim == -1:

        print(' '\*8\*layer, 'leaf (result :{})'.format(T.kind), *sep*='')

*else*:

        print(' '\*8\*layer, 'branch  划分标准: {}'.format(X\_names[T.dim]),*sep*='')

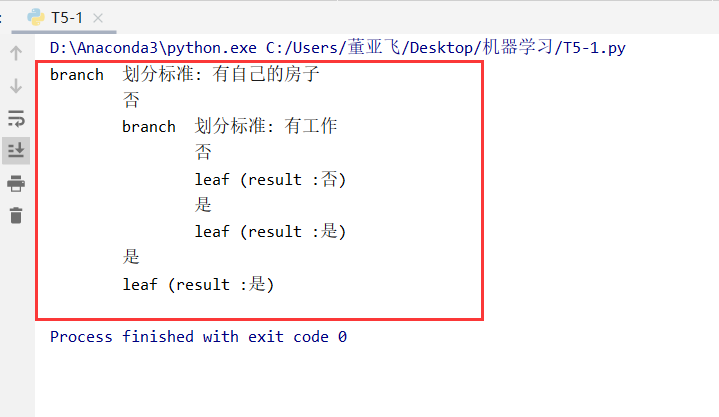
*for* d\_value, tree in T.children.items():

            print(' '\*8\*(layer+1), '{}'.format(d\_value), *sep*='')

            print\_tree(tree, layer+1)

为了清楚直观的展示出决策树的决策过程，采用树的目录表示法，规定了变量’layer’表示节点处于的不同层次来决定缩进量

1. 算法运行结果



可以看出，结果输出除了决策树的目录表示形式，可以帮助我们清楚的了解决策过程

1. 遇到的问题及解决

在最初代码实现时，在分支节点创建分支时所采用的分支均为此时的输入集中含有的对应维度的属性集合，导致生成的决策树可能存在“缺枝”现象，对部分数据无法进行决策，最后发现了这一错误，重写了对应部分的代码，完善了相应实现

#### T 5.2

问题描述：

已知如表5.2所示的训练数据，试用平方误差损失准则生成一个二叉回归树



问题解决过程：

首先，题目要求使用平方误差损失准则生成二叉回归树，需要使用CART中的回归树生成算法。决定选用python语言编程实现

代码实现过程：

1. 数据预处理

本次训练的数据量较小，故可直接在代码中写入。

1. 处理思路（模块讲解及算法流程图展示）

① 模块划分

**Data\_preprocess** : 数据预处理模块，从csv文件中读取数据并获取有用的X, y等相关信息

**Entropy**: 条件信息熵计算模块，对根据某一属性划分后的数据集求出其条件信息熵作为比较依据

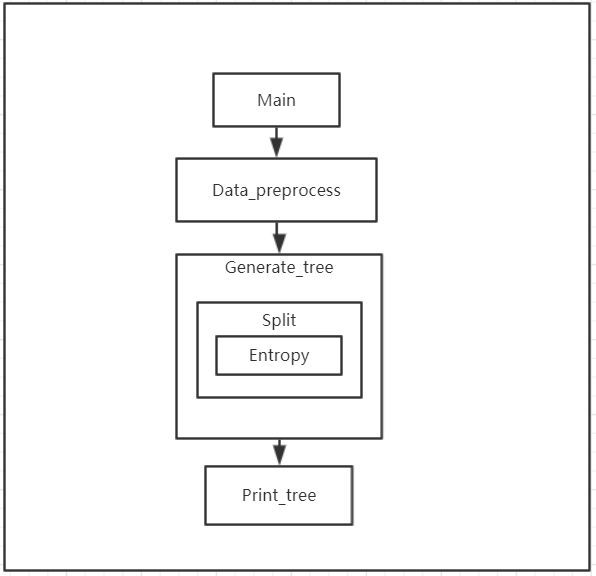
**Split**: 数据划分模块，在数据集上根据现有的可划分属性集合来对数据进行最优标准划分

**Generate\_tree:** 生成决策树模块，调用其他模块，输入初始训练数据集与初始可用划分属性集，生成训练模型决策树

**Print\_tree:** 根据所得的树结构，采用目录表示法输出所得决策树

**Main:** 主模块，调用其余模块完成训练任务

② 算法模块关系图



③ 算法重点实现部分

* 用到的Python库

*import* numpy *as* np //数据集的存储与计算

*from* collections *import* Counter //信息熵的计算与数据集类别的提取

* 决策树的数据结构

class RegressionTree:

    """定义回归树类，属性有预估值（value）| 划分值（s）| 左右子树（children）"""

    def \_\_init\_\_(*self*, *value*, *s*=float('-inf')):

        self.value = value

        self.s = s

        self.children = []

与T5-1不同，本次的输入数据x的维度为1，故决策树节点需要存储的值有value（对应特征空间的预测值）、s（切分值）、children（有两个孩子或没有，故可以不用字典）

* 平方误差计算

def square\_error(*y*):

    """求解平方误差值"""

*return* sum((y-y.mean())\*\*2)

借助于numpy.array的计算特性，便捷的完成了计算

* 数据集划分

def split(*X*,*y*,*s*):

    """根据s的值对X和y进行划分"""

    index\_a = (X <= s)

    index\_b = (X > s)

*return* X[index\_a], X[index\_b], y[index\_a], y[index\_b]

def try\_split(*X*,*y*):

    """找出针对X和y的最佳划分"""

    best\_se = float('inf')

    best\_s = float('-inf')

*#本题中提供的X值为有序排列，为了应对多种情况，假设其无序*

    sorted\_index = np.argsort(X)

*for* i in range(1, len(X)):

*if* X[sorted\_index[i]] != X[sorted\_index[i-1]]:

            s = (X[sorted\_index[i]] + X[sorted\_index[i-1]]) / 2

            X\_l, X\_r, y\_l, y\_r = split(X, y, s)

            se = square\_error(y\_l) + square\_error(y\_r)

*if* se < best\_se:

                best\_se, best\_s = se, s

*return* best\_se, best\_s

该部分与T5-1的不同在于，按指定维度进行划分时，需先将该维度的数据进行排序，对n个“不同的”属性值，找到n-1的切分值，对于每一个切分值都将原始数据集切分为左右两部分，通过遍历比较条件信息熵获取最优切分变量并返回

* 决策树生成

def generate\_RT(*X*,*y*,*depth*):

    c = y.mean()

*if* depth <= 1:

*return* RegressionTree(c)

*elif* len(Counter(X)) <= 1:

*return* RegressionTree(c)

*else*:

        se, s = try\_split(X, y)

        X\_l, X\_r, y\_l, y\_r = split(X, y, s)

        c\_l, c\_r = y\_l.mean(), y\_r.mean()

        RT = RegressionTree(c, s)

        t1 = generate\_RT(X\_l, y\_l, depth-1)

        t2 = generate\_RT(X\_r, y\_r, depth-1)

        RT.children = [t1, t2]

*return* RT

与T5-1算法大致相同，不同之处在于，每次进行递归子树生成时，只需要生成左右两颗子节点，重要的是，对于回归树的生成，需要设置最大回归深度防止过度拟合

* 输出决策树

def print\_RT(*RT*, *layer*=0):

*if* RT.s == float('-inf'):

        print(' '\*5\*layer, 'leaf(c = {})'.format(RT.value),*sep*='')

*else*:

        print(' '\*5\*layer, 'branch', *sep*='')

        print(' '\*5\*(layer+1), 'X <= {}'.format(RT.s), *sep*='')

        print\_RT(RT.children[0], layer+1)

        print(' '\*5\*(layer+1), 'X > {}'.format(RT.s), *sep*='')

        print\_RT(RT.children[1], layer+1)

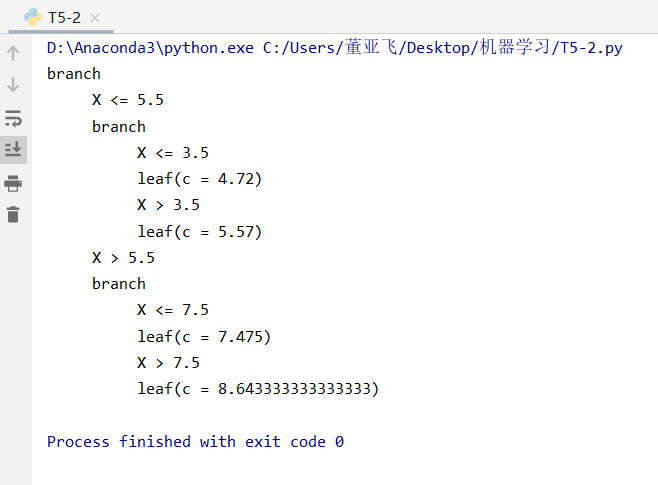
与T5-1基本相同，不做介绍

④ 算法结果输出

树深度depth = 4 (8片叶子)



树深度depth = 3（叶子数为4）



1. 遇到的问题及解决：

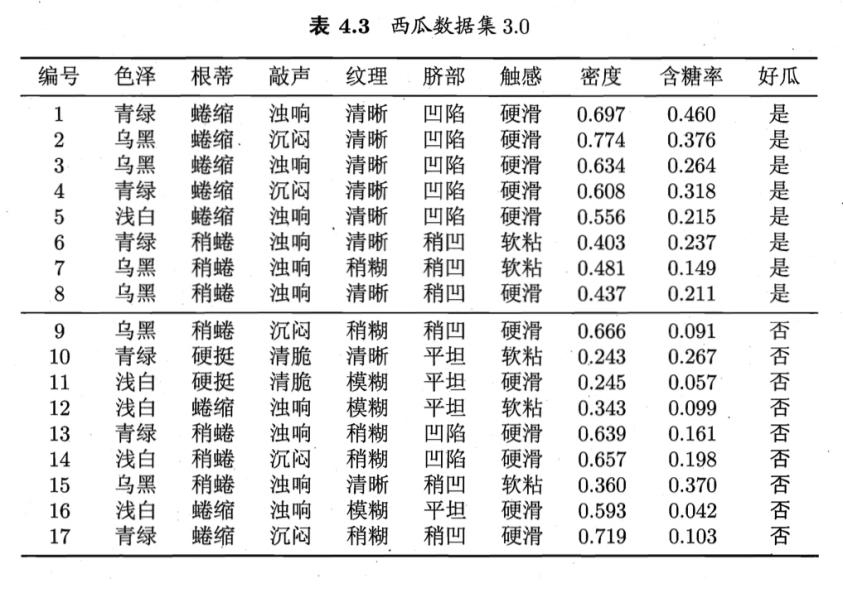
最初实现时因为忘记需要设计回归树的深度，导致产生了过拟合（每一个数据点都对应一片叶子），后来经过代码修改，增加了树的深度限制,对于本题提供的数据，将深度depth设置为3,生成4片叶子时，拟合效果较好。

### 《机器学习》（西瓜书）部分

#### T4-3

问题描述：

试编程实现基于信息熵进行划分选择的决策树算法，为表4.3中数据生成一颗决策树。



问题解决过程：

首先，题目要求以信息熵作为评估方法来生成决策树，且数据中含有连续值（维度：‘密度’，‘含糖率’）,在算法实现时需要对连续值维度做特殊处理。决定选用python语言编程实现

代码实现过程

1. 数据预处理

为使数据与代码实现分离，特从相关网站上下载了西瓜数据集3.0(题目要求数据集)，将其保存为csv格式以便于后续读取

1. 处理思路
2. 模块划分
3. 模块流程

（与T5-1、2大体相同、不再重复介绍）

1. 代码实现

* 引用的python库（与T5-1相同）

*import* numpy *as* np

*import* pandas *as* pd

*from* collections *import* Counter

*from* math *import* log

* 决策树的数据结构

class DTree:

    """构建兼容连续值与离散值的决策树"""

    def \_\_init\_\_(*self*, *kind*, *dim*, *s*=float('-inf')):

        self.kind = kind

        self.dim = dim

        self.s = s

        self.children = {}

树结构与前题大致相同，不再多做介绍

* 信息熵计算模块（略去）
* 划分模块

def split(*X*, *y*, *d*, *s*=float('-inf')):

    """对数据针对d维进行划分"""

*if* X\_names[d] not in ('密度','含糖率'):

        '''对离散值的处理'''

        X\_s, y\_s = [], []

*for* k in attributions[d]:

            X\_s.append(X[X[:, d] == k])

            y\_s.append(y[X[:, d] == k])

*return* X\_s, y\_s

*else*:

        '''对连续值的处理，进行二分类'''

        X\_s, y\_s = [], []

        X\_s.append(X[ X[:,d]<=s ])

        X\_s.append(X[ X[:,d]>s ])

        y\_s.append(y[ X[:,d]<=s ])

        y\_s.append(y[ X[:,d]>s ])

*return* X\_s, y\_s

def try\_split(*X*,*y*,*D*):

    '''对数据集X,y和属性集D,求可能划分'''

    best\_e = float('inf')   *#熵*

    best\_d = -1             *#切分维度*

    best\_s = float('-inf')  *#切分值*

*for* d in D:

*if* X\_names[d] not in ('密度','含糖率'):

            X\_s, y\_s = split(X, y, d)

            e = 0

*for* item in y\_s:

                e += item.size / len(y) \* entropy(item)

*if* e < best\_e:

                best\_d, best\_e = d, e

*else*:

            index\_sorted = np.argsort(X[:,d])

*for* i in range(1, len(X)):

*if* X[index\_sorted[i], d] != X[index\_sorted[i-1], d]:

                    s = (X[index\_sorted[i], d] + X[index\_sorted[i-1], d]) / 2

                    X\_s, y\_s = split(X,y,d,s)

                    e = y\_s[0].size / len(y) \* entropy(y\_s[0]) + y\_s[1].size / len(y) \* entropy(y\_s[1])

*if* e < best\_e:

                        best\_d, best\_e, best\_s = d, e, s

*return* best\_d, best\_e, best\_s

本题中需要注意的就是在进行划分操作时针对连续值维度与离散值维度的处理方法是不同的，需要通过分支语句对两种维度设置各自的划分方法

* 决策树生成

def generate\_Tree(*X*,*y*,*D*):

    """根据输入输出值以及可划分属性进行决策树生成"""

    kind = Counter(y).most\_common(1)[0][0]

*#数据集D中的样本为同一类*

*if* entropy(y) == 0:

*return* DTree(kind, -1)

*#数据集中的离散可用划分属性已使用完*

*if* len(D) == 2:

*return* DTree(kind, -1)

*else*:

        flag = True

*for* d in D - {6,7}:

*if* len(Counter(X[:,d])) != 1:

                flag = False

*break*

*if* flag:

*return* DTree(kind)

*else*:

            d, e, s = try\_split(X,y,D)

            X\_s, y\_s = split(X,y,d,s)

*if* X\_names[d] not in ('密度','含糖率'):

                D\_ = D.copy()

                D\_.remove(d)

                DT = DTree(kind,d)

*for* i in range(len(X\_s)):

*#遇到空样本集*

*if* X\_s[i].size == 0:

                        t = DTree(kind,-1)

*else*:

                        t = generate\_Tree(X\_s[i], y\_s[i], D\_)

                    DT.children[attributions[d][i]] = t

*return* DT

*else*:

                DT = DTree(kind, d, s)

                t0 = generate\_Tree(X\_s[0], y\_s[0], D)

                DT.children['<='+str(s)] = t0

                t1 = generate\_Tree(X\_s[1], y\_s[1], D)

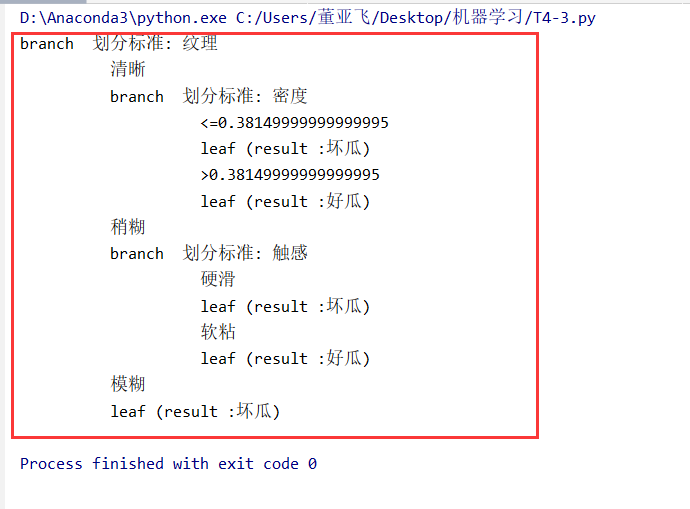
                DT.children['>'+str(s)] = t1

*return* DT

该部分的重点还是对于两种维度的不同处理方法，且对于离散维度，在划分子树时需要在属性维度集中去除已经使用过的属性，而对于连续值则不需要

* 决策树输出（与前同，略去）

1. 决策树结果输出

3、 遇到的问题及解决：

本题的难点在于如何处理好连续属性值与离散属性值的不同，针对不同的情况下也需要采用不同的处理方法及对应实现。

#### T4-4

问题描述：

试编程实现基于基尼指数进行划分选择的决策树算法，为表4.2中数据生成预剪枝、后剪枝决策树，并于未剪枝决策树进行比较。



问题解决过程：

本题中相比前面几道题增加了剪枝的操作，同时数据集也被划分成了训练集与测试集，重点需要实现剪枝算法

算法实现过程：

由于大部分实现过程前面都已经讲过，故本次主要实现与前面不相同的部分，以及预剪枝与后剪枝的实现过程

**预剪枝：**

def pre\_pruning(*X*, *y*, *X\_v*, *y\_v*, *D*):

    """生成决策树同时进行预剪枝"""

    kind = Counter(y).most\_common(1)[0][0]

*# 数据集D中样本类别相同*

*if* gini(y) == 0:

*return* DeciTree(kind, -1)

*# 可用划分属性集为空*

*if* len(D) == 0:

*return* DeciTree(kind, -1)

*# 样本在可用属性集上的取值相同*

*else*:

        flag = True

*for* d in D:

*if* len(Counter(X[:, d])) != 1:

                flag = False

*break*

*if* flag:

*return* DeciTree(kind, -1)

*else*:

            gini\_index, d = try\_split(X, y, D)

            X\_s, y\_s = split(X, y, d)

            D\_ = D.copy()

            D\_.remove(d)

            T = DeciTree(kind, d)

*# 对可能进行的划分进行预剪枝*

*# 设定变量分别表示不划分的准确数与划分后的准确度*

            unsplit\_precision, split\_precision = 0, 0

*for* i in range(len(y\_v)):

                unsplit\_precision += int(y\_v[i] == T.kind)

                d\_v = X\_v[i, d]

                find = False

*for* j in range(len(X\_s)):

*if* X\_s[j].size != 0 and X\_s[j][0, d] == d\_v:

                        find = True

*break*

*if* find:

                    split\_precision += int(y\_v[i] == Counter(y\_s[j]).most\_common(1)[0][0])

*else*:

                    split\_precision += int(y\_v[i] == T.kind)

*if* split\_precision > unsplit\_precision:

*# 划分的性能更好*

*for* i in range(len(X\_s)):

*# 遇到空样本集*

                    d\_value = attributions[d][i]

*if* X\_s[i].size == 0:

                        t = DeciTree(kind, -1)

*else*:

                        index = (X\_v[:, d] == d\_value)

                        X\_vs, y\_vs = X\_v[index], y\_v[index]

                        t = pre\_pruning(X\_s[i], y\_s[i], X\_vs, y\_vs, D\_)

                    T.children[d\_value] = t

*else*:

                T.d = -1

*return* T

（递归）这部分算法的实现需要与决策树的生成过程同步执行，需要同时输入训练数据与测试数据，关键在于当需要进行分支时：

设定变量表示“进行分支”与“不分枝”的数据正确率，并进行求解，只有当进行分支的正确率　＞ 不分枝的正确率时才会对数据进行划分，同时还要注意对那些测试数据已空的分支的处理

**后剪枝**

class DeciTree:

    """决策树结构"""

    def \_\_init\_\_(*self*, *kind*, *d*, *h*=-1):

        """叶节点的高度设置为-1"""

        self.kind = kind

        self.d = d

        self.height = h

        self.children = {}

def get\_highest\_baunch(*T*):

    """找到T中的最高分支(height值最大)"""

    flag = True

*for* t in T.children.values():

*if* t.height!=-1:

            flag = False

*break*

*#已经到达最高分支*

*if* flag:

*return* T

*#还未到达最高分支*

*else*:

        h\_height = -1

        h\_baunch = T

*for* child\_tree in T.children.values():

            temp = get\_highest\_baunch(child\_tree)

*if* temp.height > h\_height:

                h\_height = temp.height

                h\_baunch = temp

*return* h\_baunch

def post\_pruning(*X\_v*,*y\_v*,*T*):

    """根据验证数据集对决策树T进行后剪枝"""

*if* T.height == -1:

        T.height = 0

*else*:

        highest\_baunch = get\_highest\_baunch(T)

*while* highest\_baunch.height != 0:

*#剪枝前的精度*

            before\_pruning = decision\_precision(X\_v,y\_v,T)

*#剪枝后的精度*

            temp\_d = highest\_baunch.d

            highest\_baunch.d = -1

            after\_pruning = decision\_precision(X\_v,y\_v,T)

*if* after\_pruning > before\_pruning:

*pass*

*else*:

                highest\_baunch.d = temp\_d

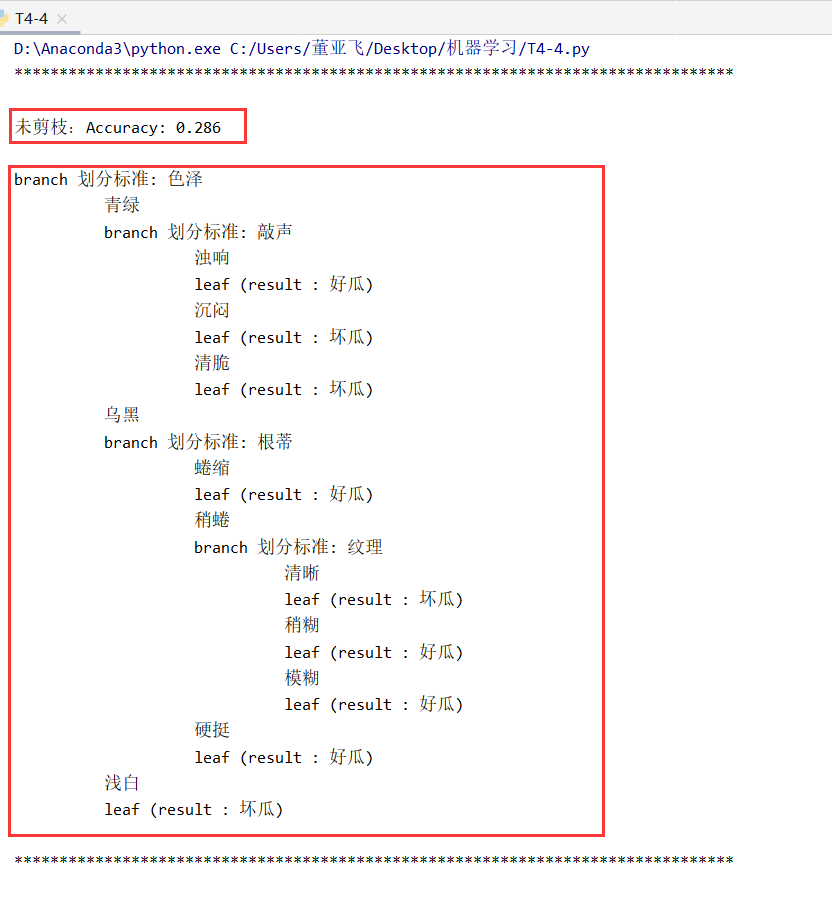
            highest\_baunch.height = -1

            highest\_baunch = get\_highest\_baunch(T)

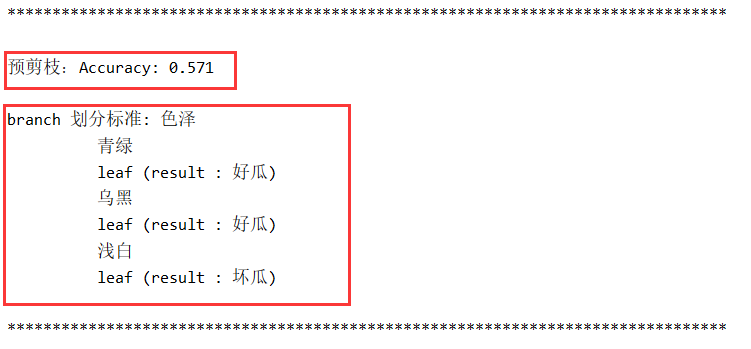
为了实现后剪枝，需要对决策树增加一个属性height(分支高度)，用以辅助确定当前需要剪枝的“最高分支”，函数get\_highest\_baunch(T)就是递归查找树T的最高分支，在post\_pruning()后剪枝函数里面，需要输入生成的未剪枝的树以及测试数据来从底部逐层向上完成剪枝

（具体实现时的剪枝做法是将节点T的d属性设为-1，代表已经是叶子节点）

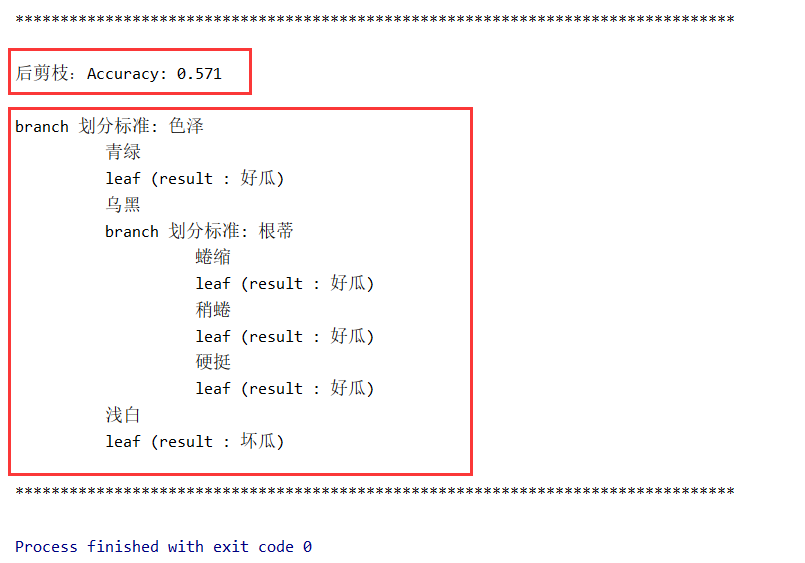
**决策树输出**



未剪枝 验证集精度：0.286



预剪枝 验证集精度：0.571



后剪枝 验证集精度：0.571

3、 遇到的问题及解决：

本题的难点在于如何处理好预剪枝及后剪枝的算法实现，对于预剪枝过程，最初由于我忽视了对于某些分支节点可能不存在验证数据的可能，导致其陷入无限递归，程序崩溃，经过良久的排查，终于找到了bug所在，成功解决。

#### T4-6

问题描述：

试选择两个UCI数据集，实现4.3、4.4的剪枝算法

（由于所使用的算法与之前大致相同，故只介绍选用的数据集情况并展示模型训练结果）

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Iris 数据集（连续属性值分类问题）

Attribute Information:

1. sepal length in cm

2. sepal width in cm

3. petal length in cm

4. petal width in cm

5. class:

-- Iris Setosa

-- Iris Versicolour

-- Iris Virginica

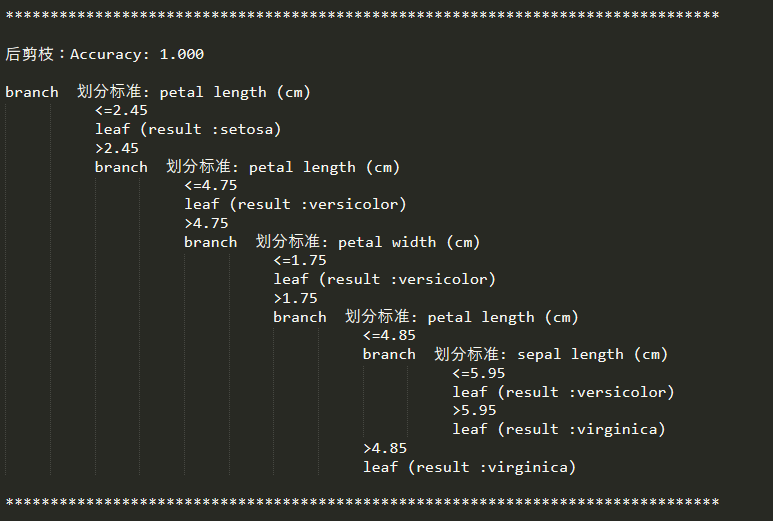
Number of Instances: 150

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

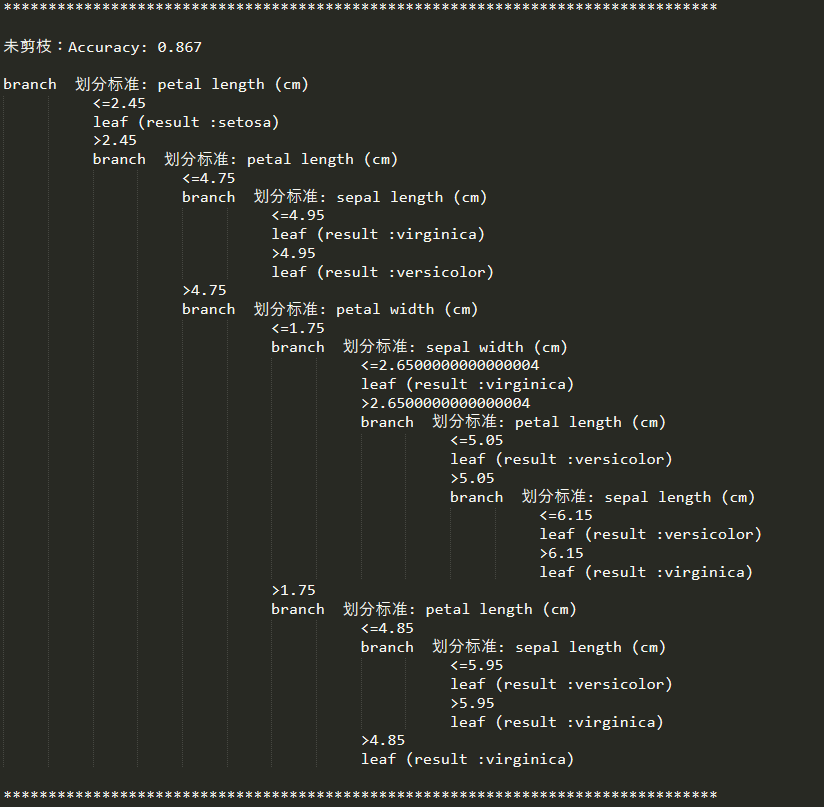
结果输出：

Iris 数据集

后剪枝：训练集精度：1



未剪枝 训练集精度：0.867



\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Car 数据集（离散属性值分类问题）

Attribute Information:

1. buying: vhigh, high, med, low.
2. maint: vhigh, high, med, low.
3. doors: 2, 3, 4, 5more.
4. persons: 2, 4, more.
5. lug\_boot: small, med, big.
6. safety: low, med, high.
7. class

-- unacc

-- acc

-- good

-- vgood

Number of Instances: 1728

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

（由于生成的决策树过大，所以可以直接跳转观看）

[未剪枝决策树](#未剪枝)

[预剪枝决策树](#预剪枝)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

未剪枝：Accuracy: 0.945

branch 划分标准: safety

med

branch 划分标准: persons

more

branch 划分标准: lug\_boot

med

branch 划分标准: buying

vhigh

branch 划分标准: maint

low

branch 划分标准: doors

4

leaf (result : acc)

5more

leaf (result : acc)

3

leaf (result : acc)

2

leaf (result : unacc)

med

branch 划分标准: doors

4

leaf (result : acc)

5more

leaf (result : acc)

3

leaf (result : acc)

2

leaf (result : unacc)

vhigh

leaf (result : unacc)

high

leaf (result : unacc)

med

branch 划分标准: maint

low

branch 划分标准: doors

4

leaf (result : good)

5more

leaf (result : good)

3

leaf (result : good)

2

leaf (result : acc)

med

leaf (result : acc)

vhigh

branch 划分标准: doors

4

leaf (result : acc)

5more

leaf (result : acc)

3

leaf (result : acc)

2

leaf (result : unacc)

high

branch 划分标准: doors

4

leaf (result : acc)

5more

leaf (result : acc)

3

leaf (result : acc)

2

leaf (result : unacc)

low

branch 划分标准: maint

low

branch 划分标准: doors

4

leaf (result : good)

5more

leaf (result : good)

3

leaf (result : good)

2

leaf (result : acc)

med

leaf (result : good)

vhigh

branch 划分标准: doors

4

leaf (result : acc)

5more

leaf (result : acc)

3

leaf (result : acc)

2

leaf (result : unacc)

high

leaf (result : acc)

high

branch 划分标准: doors

4

leaf (result : acc)

5more

branch 划分标准: maint

low

leaf (result : acc)

med

leaf (result : acc)

vhigh

leaf (result : unacc)

high

leaf (result : acc)

3

leaf (result : acc)

2

leaf (result : unacc)

big

branch 划分标准: maint

low

branch 划分标准: buying

vhigh

leaf (result : acc)

med

leaf (result : good)

low

leaf (result : good)

high

leaf (result : acc)

med

branch 划分标准: buying

vhigh

leaf (result : acc)

med

leaf (result : acc)

low

leaf (result : good)

high

leaf (result : acc)

vhigh

branch 划分标准: buying

vhigh

leaf (result : unacc)

med

leaf (result : acc)

low

leaf (result : acc)

high

leaf (result : unacc)

high

branch 划分标准: buying

vhigh

leaf (result : unacc)

med

leaf (result : acc)

low

leaf (result : acc)

high

leaf (result : acc)

small

branch 划分标准: buying

vhigh

leaf (result : unacc)

med

branch 划分标准: maint

low

branch 划分标准: doors

4

leaf (result : acc)

5more

leaf (result : acc)

3

leaf (result : acc)

2

leaf (result : unacc)

med

branch 划分标准: doors

4

leaf (result : acc)

5more

leaf (result : acc)

3

leaf (result : acc)

2

leaf (result : unacc)

vhigh

leaf (result : unacc)

high

leaf (result : unacc)

low

branch 划分标准: maint

low

branch 划分标准: doors

4

leaf (result : acc)

5more

leaf (result : acc)

3

leaf (result : acc)

2

leaf (result : unacc)

med

branch 划分标准: doors

4

leaf (result : acc)

5more

leaf (result : acc)

3

leaf (result : acc)

2

leaf (result : unacc)

vhigh

leaf (result : unacc)

high

branch 划分标准: doors

4

leaf (result : acc)

5more

leaf (result : acc)

3

leaf (result : acc)

2

leaf (result : unacc)

high

leaf (result : unacc)

2

leaf (result : unacc)

4

branch 划分标准: lug\_boot

med

branch 划分标准: doors

4

branch 划分标准: maint

low

leaf (result : good)

med

branch 划分标准: buying

vhigh

leaf (result : acc)

med

leaf (result : acc)

low

leaf (result : good)

high

leaf (result : acc)

vhigh

branch 划分标准: buying

vhigh

leaf (result : unacc)

med

leaf (result : acc)

low

leaf (result : acc)

high

leaf (result : unacc)

high

branch 划分标准: buying

vhigh

leaf (result : unacc)

med

leaf (result : acc)

low

leaf (result : acc)

high

leaf (result : acc)

5more

branch 划分标准: buying

vhigh

branch 划分标准: maint

low

leaf (result : acc)

med

leaf (result : acc)

vhigh

leaf (result : unacc)

high

leaf (result : unacc)

med

branch 划分标准: maint

low

leaf (result : good)

med

leaf (result : acc)

vhigh

leaf (result : acc)

high

leaf (result : acc)

low

leaf (result : acc)

high

leaf (result : acc)

3

branch 划分标准: buying

vhigh

leaf (result : unacc)

med

branch 划分标准: maint

low

leaf (result : acc)

med

leaf (result : acc)

vhigh

leaf (result : unacc)

high

leaf (result : unacc)

low

branch 划分标准: maint

low

leaf (result : acc)

med

leaf (result : acc)

vhigh

leaf (result : unacc)

high

leaf (result : acc)

high

leaf (result : unacc)

2

branch 划分标准: buying

vhigh

leaf (result : unacc)

med

branch 划分标准: maint

low

leaf (result : acc)

med

leaf (result : acc)

vhigh

leaf (result : acc)

high

leaf (result : unacc)

low

branch 划分标准: maint

low

leaf (result : acc)

med

leaf (result : acc)

vhigh

leaf (result : unacc)

high

leaf (result : acc)

high

leaf (result : unacc)

big

branch 划分标准: buying

vhigh

branch 划分标准: maint

low

leaf (result : acc)

med

leaf (result : acc)

vhigh

leaf (result : unacc)

high

leaf (result : unacc)

med

branch 划分标准: maint

low

leaf (result : good)

med

leaf (result : acc)

vhigh

leaf (result : acc)

high

leaf (result : acc)

low

branch 划分标准: maint

low

leaf (result : good)

med

leaf (result : good)

vhigh

leaf (result : acc)

high

leaf (result : acc)

high

branch 划分标准: maint

low

leaf (result : acc)

med

leaf (result : acc)

vhigh

leaf (result : unacc)

high

leaf (result : acc)

small

branch 划分标准: buying

vhigh

leaf (result : unacc)

med

branch 划分标准: maint

low

leaf (result : acc)

med

leaf (result : acc)

vhigh

leaf (result : unacc)

high

leaf (result : unacc)

low

branch 划分标准: maint

low

leaf (result : acc)

med

leaf (result : acc)

vhigh

leaf (result : unacc)

high

leaf (result : acc)

high

leaf (result : unacc)

low

leaf (result : unacc)

high

branch 划分标准: persons

more

branch 划分标准: buying

vhigh

branch 划分标准: maint

low

branch 划分标准: doors

4

leaf (result : acc)

5more

leaf (result : acc)

3

leaf (result : acc)

2

branch 划分标准: lug\_boot

med

leaf (result : acc)

big

leaf (result : acc)

small

leaf (result : unacc)

med

branch 划分标准: lug\_boot

med

leaf (result : acc)

big

leaf (result : acc)

small

branch 划分标准: doors

4

leaf (result : acc)

5more

leaf (result : unacc)

3

leaf (result : unacc)

2

leaf (result : unacc)

vhigh

leaf (result : unacc)

high

leaf (result : unacc)

med

branch 划分标准: maint

low

branch 划分标准: lug\_boot

med

branch 划分标准: doors

4

leaf (result : vgood)

5more

leaf (result : vgood)

3

leaf (result : vgood)

2

leaf (result : good)

big

leaf (result : vgood)

small

leaf (result : good)

med

branch 划分标准: lug\_boot

med

branch 划分标准: doors

4

leaf (result : vgood)

5more

leaf (result : vgood)

3

leaf (result : vgood)

2

leaf (result : acc)

big

leaf (result : vgood)

small

branch 划分标准: doors

4

leaf (result : acc)

5more

leaf (result : acc)

3

leaf (result : acc)

2

leaf (result : unacc)

vhigh

branch 划分标准: doors

4

leaf (result : acc)

5more

leaf (result : acc)

3

leaf (result : acc)

2

branch 划分标准: lug\_boot

med

leaf (result : unacc)

big

leaf (result : acc)

small

leaf (result : unacc)

high

leaf (result : acc)

low

branch 划分标准: maint

low

branch 划分标准: lug\_boot

med

leaf (result : vgood)

big

leaf (result : vgood)

small

branch 划分标准: doors

4

leaf (result : good)

5more

leaf (result : good)

3

leaf (result : good)

2

leaf (result : unacc)

med

branch 划分标准: lug\_boot

med

branch 划分标准: doors

4

leaf (result : vgood)

5more

leaf (result : vgood)

3

leaf (result : vgood)

2

leaf (result : good)

big

leaf (result : vgood)

small

branch 划分标准: doors

4

leaf (result : good)

5more

leaf (result : good)

3

leaf (result : good)

2

leaf (result : unacc)

vhigh

branch 划分标准: doors

4

leaf (result : acc)

5more

leaf (result : acc)

3

leaf (result : acc)

2

branch 划分标准: lug\_boot

med

leaf (result : acc)

big

leaf (result : acc)

small

leaf (result : unacc)

high

branch 划分标准: lug\_boot

med

leaf (result : vgood)

big

leaf (result : vgood)

small

branch 划分标准: doors

4

leaf (result : acc)

5more

leaf (result : acc)

3

leaf (result : acc)

2

leaf (result : unacc)

high

branch 划分标准: maint

low

branch 划分标准: doors

4

leaf (result : acc)

5more

leaf (result : acc)

3

leaf (result : acc)

2

branch 划分标准: lug\_boot

med

leaf (result : acc)

big

leaf (result : acc)

small

leaf (result : unacc)

med

branch 划分标准: doors

4

leaf (result : acc)

5more

leaf (result : acc)

3

leaf (result : acc)

2

branch 划分标准: lug\_boot

med

leaf (result : acc)

big

leaf (result : acc)

small

leaf (result : unacc)

vhigh

leaf (result : unacc)

high

leaf (result : acc)

2

leaf (result : unacc)

4

branch 划分标准: buying

vhigh

branch 划分标准: maint

low

leaf (result : acc)

med

leaf (result : acc)

vhigh

leaf (result : unacc)

high

leaf (result : unacc)

med

branch 划分标准: maint

low

branch 划分标准: lug\_boot

med

branch 划分标准: doors

4

leaf (result : vgood)

5more

leaf (result : vgood)

3

leaf (result : good)

2

leaf (result : good)

big

leaf (result : vgood)

small

leaf (result : good)

med

branch 划分标准: lug\_boot

med

branch 划分标准: doors

4

leaf (result : vgood)

5more

leaf (result : vgood)

3

leaf (result : acc)

2

leaf (result : vgood)

big

leaf (result : vgood)

small

leaf (result : acc)

vhigh

leaf (result : acc)

high

leaf (result : acc)

low

branch 划分标准: maint

low

branch 划分标准: lug\_boot

med

branch 划分标准: doors

4

leaf (result : vgood)

5more

leaf (result : vgood)

3

leaf (result : good)

2

leaf (result : good)

big

leaf (result : vgood)

small

leaf (result : good)

med

branch 划分标准: lug\_boot

med

branch 划分标准: doors

4

leaf (result : vgood)

5more

leaf (result : vgood)

3

leaf (result : good)

2

leaf (result : good)

big

leaf (result : vgood)

small

leaf (result : good)

vhigh

leaf (result : acc)

high

branch 划分标准: lug\_boot

med

branch 划分标准: doors

4

leaf (result : vgood)

5more

leaf (result : acc)

3

leaf (result : acc)

2

leaf (result : acc)

big

leaf (result : vgood)

small

leaf (result : acc)

high

branch 划分标准: maint

low

leaf (result : acc)

med

leaf (result : acc)

vhigh

leaf (result : unacc)

high

leaf (result : acc)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

[点击返回](#返回)

预剪枝：Accuracy: 0.829

branch 划分标准: safety

med

branch 划分标准: persons

more

branch 划分标准: lug\_boot

med

leaf (result : acc)

big

branch 划分标准: maint

low

branch 划分标准: buying

vhigh

leaf (result : acc)

med

leaf (result : good)

low

leaf (result : good)

high

leaf (result : acc)

med

branch 划分标准: buying

vhigh

leaf (result : acc)

med

leaf (result : acc)

low

leaf (result : good)

high

leaf (result : acc)

vhigh

branch 划分标准: buying

vhigh

leaf (result : unacc)

med

leaf (result : acc)

low

leaf (result : acc)

high

leaf (result : unacc)

high

branch 划分标准: buying

vhigh

leaf (result : unacc)

med

leaf (result : acc)

low

leaf (result : acc)

high

leaf (result : acc)

small

branch 划分标准: buying

vhigh

leaf (result : unacc)

med

branch 划分标准: maint

low

branch 划分标准: doors

4

leaf (result : acc)

5more

leaf (result : acc)

3

leaf (result : acc)

2

leaf (result : unacc)

med

branch 划分标准: doors

4

leaf (result : acc)

5more

leaf (result : acc)

3

leaf (result : acc)

2

leaf (result : unacc)

vhigh

leaf (result : unacc)

high

leaf (result : unacc)

low

branch 划分标准: maint

low

branch 划分标准: doors

4

leaf (result : acc)

5more

leaf (result : acc)

3

leaf (result : acc)

2

leaf (result : unacc)

med

branch 划分标准: doors

4

leaf (result : acc)

5more

leaf (result : acc)

3

leaf (result : acc)

2

leaf (result : unacc)

vhigh

leaf (result : unacc)

high

branch 划分标准: doors

4

leaf (result : acc)

5more

leaf (result : acc)

3

leaf (result : acc)

2

leaf (result : unacc)

high

leaf (result : unacc)

2

leaf (result : unacc)

4

leaf (result : acc)

low

leaf (result : unacc)

high

branch 划分标准: persons

more

leaf (result : acc)

2

leaf (result : unacc)

4

leaf (result : acc)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1. 实验问题及解决

经过这次试验，我最大的收获是对于预剪枝，由于采用的是贪心算法，所以剪枝后的决策树党的精度值可能有所下降。

## 实验收获

通过本次编程实验，我对决策树相关算法有了更直观与细致的了解，对于算法中的一些要点难点的代码实现也有了直观的了解。 同时我对于不同评价标准对于训练所得的决策树模型的影响有了初步的了解。

除此之外，通过亲自动手实现相关算法，我对于使用Python语言进行数据分析与机器学习算法的实现方法有了更多的了解，编程动手能力有所提高。