**作业4.4**

1. **实验要求**

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，并与未剪枝决策树进行比较。

1. **实验原理**

与4.3相比，在绘制决策树时只需要将信息熵的那部分换成基尼系数即可。不同之处在于此题还需要画出预剪枝和后剪枝，预剪枝是在决策前进行判别下一步是否可以带来性能的提升，而去剪枝则是在划分好决策树之后判断去除当前属性划分后能否带来性能的提升。

1. **实验过程**

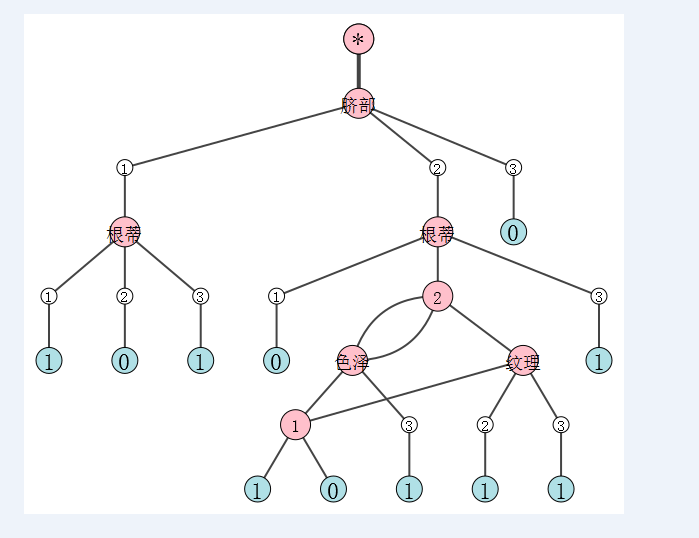
分别定义未剪枝、预剪枝和后剪枝的决策树，过程基本与4.3类似，代码如下：

**import** numpy **as** np  
**import** igraph **as** ig  
**from** collections **import** Counter  
**import** random, copy  
  
  
**def** all\_class(D): *# 判断D中类的个数* **return** len(np.unique(D[:, -1])) *# np.unique用以生成具有不重复元素的数组***def** diff\_in\_attr(D, A\_code): *# 判断D中样本在当前A\_code上是否存在属性取值不同，即各属性只有一个取值* cond = **False  
 for** i **in** A\_code: *# continuous args are excluded* **if** len(np.unique(D[:, i])) != 1:  
 cond = **True  
 break  
 return** cond  
  
  
**def** major\_class(D): *# 票选出D中样本数最多的类  
 # 本书所给图4.6是利用信息增益划分得到的，在进行预剪枝时，将脐部=稍凹\  
 # 判定为好瓜和坏瓜具有同样验证集精度和泛化误差，但若直接用22行进行计算\  
 # 取第一项，则没有考虑到训练集含偶数个例子时，可能出现上述情况，为了和书\  
 # 上一致，强行让这种情况下的分类取为1，即好瓜* c = Counter(D[:, -1]).most\_common()  
 **if** len(c) == 1:  
 **return** c[0][0]  
 **elif** c[0][1] == c[1][1]:  
 **return** 1  
 **else**:  
 **return** c[0][0]  
  
  
**def** min\_gini(D, A\_code):  
 N = len(D) *# 当前样本数* dict\_class = {} *# 用字典保存类及其所在样本,键为类型，值为类型所含样本* **for** i **in** range(N):  
 **if** D[i, -1] **not in** dict\_class.keys(): *# 为字典添加新类* dict\_class[D[i, -1]] = []  
 dict\_class[D[i, -1]].append(int(D[i, 0]))  
 **else**:  
 dict\_class[D[i, -1]].append(int(D[i, 0]))  
 Gini\_D\_A = {} *# 用字典保存在某一属性下的属性值及其所在样本,键为属性取值，值为对应样本* **for** a **in** A\_code:  
 *# A中的离散属性,在后期的迭代中，属性a可能会变得不连续* dict\_attr = {}  
 **for** i **in** range(N):  
 **if** D[i, a] **not in** dict\_attr.keys(): *# 为字典添加新的属性取值* dict\_attr[D[i, a]] = []  
 dict\_attr[D[i, a]].append(int(D[i, 0]))  
 **else**:  
 dict\_attr[D[i, a]].append(int(D[i, 0]))  
 *# 对Gini\_index(D,a)变形发现，在只有好瓜和坏瓜这两类时，基尼指数最小等价于  
 # sigma(v=1到V)求和：(|Dv+| \* |Dv-|) / |Dv|最小* Gini\_D\_A[(a,)] = 0  
 **for** av, v **in** dict\_attr.items():  
 m = len(v) *# m为当前属性取值的样本总数,如A\_a0包含的样本总数* x2 = len(set(v) & set(dict\_class[1.0]))  
 x1 = m - x2  
 Gini\_D\_A[(a,)] += x1 \* x2 / m  
 Gini\_D\_A\_list = sorted(Gini\_D\_A.items(), key=**lambda** a: a[1])  
 best = Gini\_D\_A\_list[0][0]  
 dict\_attr = {}  
 best = int(best[0])  
 **for** i **in** range(N):  
 **if** D[i, best] **not in** dict\_attr.keys():  
 dict\_attr[D[i, best]] = []  
 dict\_attr[D[i, best]].append(int(D[i, 0]))  
 **else**:  
 dict\_attr[D[i, best]].append(int(D[i, 0]))  
 *# 返回对应的属性序号（绝对序号），并返回此属性下对应的取值和所包含的实例的id* **return** best, dict\_attr  
  
  
**def** prim\_Tree\_Generate(D, A\_code, full\_D):  
 *# 生成未剪枝决策树* **if** all\_class(D) == 1: *# case1* **return** D[0, -1]  
 **if** (**not** A\_code) **or** (**not** diff\_in\_attr(D, A\_code)): *# case2* **return** major\_class(D)  
 a, di = min\_gini(D, A\_code)  
 a\_name = **'%s%s'** % (A[a], major\_class(D))  
 np\_tree = {a\_name: {}}  
 all\_a = np.unique(full\_D[:, a])  
 new\_A\_code = A\_code[:]  
 new\_A\_code.remove(a)  
 **for** item **in** all\_a:  
 **if** item **not in** di.keys():  
 di[item] = []  
  
 **for** av, Dv **in** di.items():  
 **if** Dv:  
 np\_tree[a\_name][av] = prim\_Tree\_Generate(full\_D[Dv, :], new\_A\_code, full\_D)  
 **else**: *# case3* np\_tree[a\_name][av] = major\_class(D)  
 **return** np\_tree  
  
  
**def** pre\_Tree\_Generate(A\_code, train\_D, test\_D, full\_D, pre\_correct):  
 *# 生成预剪枝决策树  
 # pre\_correct代表此节点对应的同父的子节点的划分前验证集精度  
 # 如脐部（heart）取值为1,2,3的节点都应该具有同样的pre\_correct  
 # 对于根节点，其pre\_correct需要在函数外提前用major\_class()获得，作为参数传入* **if** all\_class(train\_D) == 1: *# case1* **return** train\_D[0, -1]  
 **if** (**not** A\_code) **or** (**not** diff\_in\_attr(train\_D, A\_code)): *# case2* **return** major\_class(train\_D)  
 a, di = min\_gini(train\_D, A\_code)  
 all\_a = np.unique(full\_D[:, a])  
 **for** item **in** all\_a:  
 **if** item **not in** di.keys():  
 di[item] = []  
  
 *# 比较此节点划分前后的泛化能力* n, post\_correct = len(test\_D), 0  
 node\_label = major\_class(train\_D)  
 judge, test\_D\_v = {}, {}  
 **for** av, Dv **in** di.items():  
 **if not** Dv:  
 judge[av] = major\_class(train\_D) *# 节点的多数类* **else**:  
 judge[av] = major\_class(full\_D[Dv, :]) *# 各子集的多数类* **for** i **in** range(n):  
 **if** judge[test\_D[i, a]] == test\_D[i, -1]:  
 post\_correct += 1 *# 划分节点后的正确率* **if** test\_D[i, a] **not in** test\_D\_v.keys():  
 test\_D\_v[test\_D[i, a]] = []  
 test\_D\_v[test\_D[i, a]].append(int(test\_D[i, 0]))  
 **else**:  
 test\_D\_v[test\_D[i, a]].append(int(test\_D[i, 0]))  
 post\_correct /= n  
 **if** post\_correct <= pre\_correct: *# 决定是否剪枝* **return** node\_label  
 **else**:  
 a\_name = **'%s%s'** % (A[a], major\_class(train\_D))  
 tree = {a\_name: {}}  
 new\_A\_code = A\_code[:]  
 new\_A\_code.remove(a)  
 **for** av, Dv **in** di.items():  
 **if** Dv:  
 tdv = test\_D\_v[av]  
 tree[a\_name][av] = pre\_Tree\_Generate(new\_A\_code, full\_D[Dv, :], full\_D[tdv, :], full\_D, post\_correct)  
 **else**: *# case3* tree[a\_name][av] = major\_class(train\_D)  
 **return** tree  
  
  
**def** d\_tree(tree, test\_D, anti\_A, full\_D):  
 *# 提供给tree\_update()函数进行调用，为后剪枝函数1  
 # 返回未剪枝决策树tree中，各节点包含的测试集test\_D内实例形成的字典，形式与决策树类似，具体如下：  
 # {'脐部1': {1: {'根蒂1': {1: [3, 4], 2: [12], 3: []}}, 2: {'根蒂1': {1: [], \  
 # 2: {'色泽1': {1: [], 2: {'纹理1': {1: [7], 2: [8], 3: []}}, 3: []}}, 3: []}}, 3: [10, 11]}}* dimension = len(test\_D.shape)  
 **if** type(tree).\_\_name\_\_ != **'dict'**:  
 id = []  
 **if** dimension == 1:  
 id.append(test\_D[0])  
 **for** i **in** range(len(test\_D)):  
 id.append(test\_D[i, 0])  
 **return** id  
 a, di = list(tree.items())[0]  
 data\_tree = {a: {}}  
 acode = anti\_A[a[:-1]]  
 **for** av, dv **in** di.items():  
 **if** type(dv).\_\_name\_\_ == **'int'**:  
 data\_tree[a][av] = []  
 **if** dimension == 1:  
 **if** test\_D[acode] == av:  
 data\_tree[a][av].append(test\_D[0])  
 **else**:  
 **for** row **in** range(len(test\_D)):  
 **if** test\_D[row, acode] == av:  
 data\_tree[a][av].append(test\_D[row, 0])  
 **else**:  
 DV = []  
 **if** dimension == 1:  
 **if** test\_D[acode] == av:  
 DV.append(test\_D[0])  
 **else**:  
 **for** row **in** range(len(test\_D)):  
 **if** test\_D[row, acode] == av:  
 DV.append(test\_D[row, 0])  
 data\_tree[a][av] = d\_tree(dv, full\_D[DV, :], anti\_A, full\_D)  
  
 **return** data\_tree  
  
  
**def** tree\_update(data\_tree, edit\_tree, full\_D, pa\_tree=**None**, pa\_d\_tree=**None**, pa\_a=**None**):  
 *# 提供给post\_tree()函数进行调用，为后剪枝函数2  
 # 利用当前决策树和各节点所含测试例进行后剪枝，具体就是对当前树中所有叶节点判断是否剪枝，若是，则\  
 # 一次性将所有满足剪枝条件的叶节点进行剪枝处理，就像剥洋葱一样，一次剥掉一层而不管这一层的面积  
 # data\_tree为测试集在节点中分布的字典，edit\_tree为要进行单次剪枝的决策树  
 # pa\_tree代表包含当前edit\_tree的父节点的字典，pa\_d\_tree代表包含\  
 # 当前data\_tree的父节点的字典，pa\_a为指向当前edit\_tree的键，\  
 # 例如：edit\_tree={'色泽1': {1: 1, 2: 1, 3: 1}},\  
 # 则对应的pa\_tree={'根蒂1': {1: 0, 2: {'色泽1': {1: 1, 2: 1, 3: 1}}, 3: 1}}, \  
 # pa\_a=2, pa\_d\_tree同理* a = list(edit\_tree.keys())[0]  
 d = data\_tree[a]  
 *# print(d)* flag, node\_data = [], []  
 **for** datum **in** list(d.values()):  
 **if** type(datum).\_\_name\_\_ == **'dict'**:  
 flag.append(datum)  
 **break  
 if not** flag:  
 pre, post = 0, 0  
 **for** av, dv **in** d.items():  
 **if** dv:  
 **for** i **in** dv:  
 node\_data.append(i)  
 **if** full\_D[i, -1] == edit\_tree[a][av]:  
 pre += 1  
 **if** full\_D[i, -1] == int(a[-1]):  
 post += 1  
  
 **if** post > pre **and** pa\_tree:  
 *# 取>=时，根据奥卡姆剃刀准则，性能不下降就剪枝  
 # 剪枝最后的决策树为{'脐部1': {1: {'根蒂1': {1: 1, 2: 0, 3: 1}}, 2: 1, 3: 0}}  
 # 相应地，各节点包含的测试集实例为\  
 # {'脐部1': {1: {'根蒂1': {1: [3, 4], 2: [12], 3: []}}, 2: [7, 8], 3: [10, 11]}}  
 # 取>时，与书上的例子一致，性能提升才剪枝  
 # 最终决策树为{'脐部1': {1: {'根蒂1': {1: 1, 2: 0, 3: 1}}, \  
 # 2: {'根蒂1': {1: 0, 2: {'色泽1': {1: 1, 2: 1, 3: 1}}, 3: 1}}, 3: 0}}  
 # 各节点包含的测试集实例\  
 # {'脐部1': {1: {'根蒂1': {1: [3, 4], 2: [12], 3: []}}, 2: {'根蒂1': {1: [], \  
 # 2: {'色泽1': {1: [], 2: [7, 8], 3: []}}, 3: []}}, 3: [10, 11]}}* pa\_tree[pa\_a] = int(a[-1])  
 pa\_d\_tree[pa\_a] = node\_data  
 **else**:  
 **for** av, dv **in** edit\_tree[a].items():  
 new\_d\_tree = data\_tree[a][av]  
 **if** type(dv).\_\_name\_\_ == **'dict' and** new\_d\_tree:  
 tree\_update(new\_d\_tree, dv, full\_D, pa\_tree=edit\_tree[a], pa\_d\_tree=data\_tree[a], pa\_a=av)  
 **return** data\_tree, edit\_tree  
  
  
**def** post\_tree(data\_tree, edit\_tree, D):  
 *# 返回最终的后剪枝处理完成的决策树，为后剪枝函数3  
 # 之所以要多次调用tree\_update()，是因为tree\_update()剪枝后形成的新叶节点仍然满足剪枝条件* old\_l = copy.deepcopy(edit\_tree)  
 dd, ll = tree\_update(data\_tree, edit\_tree, D)  
 **if** old\_l != ll:  
 **return** post\_tree(dd, ll, D)  
 **else**:  
 **return** ll  
  
  
**def** Tree\_draw(tree\_item, g, node=0):  
 *# 获取树节点的拓扑关系并绘图* rand = random.randint(100, 999)  
 attr, cond = tree\_item  
 u\_attr = **'%s\_%s'** % (attr, rand) *# 保证各节点name的独特性，在后面用来查找此节点* g.add\_vertex(u\_attr)  
 new\_node = g.vs.find(name=u\_attr).index  
 this\_label = str(attr)  
 **if** len(this\_label) > 1: *# 如果节点名长度大于1，则类似“根蒂1”，应该将其显示为“根蒂”* this\_label = this\_label[:-1]  
 g.vs[new\_node][**'label'**] = this\_label  
 g.add\_edge(node, new\_node)  
 **if** type(cond).\_\_name\_\_ == **'dict'**:  
 **for** item **in** list(cond.items()):  
 Tree\_draw(item, g, new\_node)  
 **else**:  
 u\_cond = **'%s\_%s'** % (rand, cond)  
 g.add\_vertex(u\_cond)  
 end\_node = g.vs.find(name=u\_cond).index  
 g.vs[end\_node][**'label'**] = str(int(cond))  
 g.add\_edge(new\_node, end\_node)  
 **return** g  
  
  
*# ###数据准备及预处理###*D = np.array([  
 [0, 1, 1, 1, 1, 1, 1, 1],  
 [1, 2, 1, 2, 1, 1, 1, 1],  
 [2, 2, 1, 1, 1, 1, 1, 1],  
 [3, 1, 1, 2, 1, 1, 1, 1],  
 [4, 3, 1, 1, 1, 1, 1, 1],  
 [5, 1, 2, 1, 1, 2, 2, 1],  
 [6, 2, 2, 1, 2, 2, 2, 1],  
 [7, 2, 2, 1, 1, 2, 1, 1],  
 [8, 2, 2, 2, 2, 2, 1, 0],  
 [9, 1, 3, 3, 1, 3, 2, 0],  
 [10, 3, 3, 3, 3, 3, 1, 0],  
 [11, 3, 1, 1, 3, 3, 2, 0],  
 [12, 1, 2, 1, 2, 1, 1, 0],  
 [13, 3, 2, 2, 2, 1, 1, 0],  
 [14, 2, 2, 1, 1, 2, 2, 0],  
 [15, 3, 1, 1, 3, 3, 1, 0],  
 [16, 1, 1, 2, 2, 2, 1, 0]])  
train\_D = D[[0, 1, 2, 5, 6, 9, 13, 14, 15, 16], :]  
test\_D = D[[3, 4, 7, 8, 10, 11, 12], :]  
A = {0: **'id'**, 1: **'色泽'**, 2: **'根蒂'**, 3: **'敲声'**, 4: **'纹理'**, 5: **'脐部'**, 6: **'触感'**, 7: **'好瓜'**}  
A\_code = list(range(1, len(A) - 1)) *# A\_code = [1, 2, 3, 4, 5, 6]*anti\_A = {**'id'**: 0, **'色泽'**: 1, **'根蒂'**: 2, **'敲声'**: 3, **'纹理'**: 4, **'脐部'**: 5, **'触感'**: 6, **'好瓜'**: 7}  
*# ###################  
  
# ########未剪枝#######*tree = prim\_Tree\_Generate(train\_D, A\_code, D)  
*# ####################  
  
# ########预剪枝#######  
# n,node\_label,init\_correct = len(test\_D),major\_class(train\_D),0  
# for i in range(n):  
# if test\_D[i, -1] == node\_label:  
# init\_correct += 1 # 划分节点前的正确个数  
# init\_correct /= n # 初始验证集精度  
# tree = pre\_Tree\_Generate(A\_code,train\_D,test\_D,D,init\_correct)  
# ###################  
  
# #######后剪枝#######  
# edit\_tree = prim\_Tree\_Generate(train\_D,A\_code,D)  
# data\_tree = d\_tree(edit\_tree,test\_D,anti\_A,D)  
# tree = post\_tree(data\_tree, edit\_tree, D)  
# ###################*print(tree)  
  
*# #########生成决策树图#########*init\_items = list(tree.items())[0]  
g = ig.Graph()  
g.add\_vertex(**'Source'**)  
g.vs[0][**'label'**] = **'\*'**s = Tree\_draw(init\_items, g, node=0)  
  
  
*# ###########################  
  
# ########修饰决策树图表########***def** dyer(x):  
 **if** x == 2:  
 **return 'white'  
 elif** x == 1:  
 **return 'powderblue'  
 else**:  
 **return 'pink'**dye = list(map(dyer, s.vs.degree()))  
label\_size\_map = {**'white'**: 15, **'pink'**: 18, **'powderblue'**: 24}  
v\_size\_map = {**'white'**: 16, **'pink'**: 30, **'powderblue'**: 26}  
label\_size = [label\_size\_map[i] **for** i **in** dye]  
v\_size = [v\_size\_map[i] **for** i **in** dye]  
edge\_width = [2] \* (len(dye) - 1)  
edge\_width[0] = 4  
my\_lay = g.layout\_reingold\_tilford(root=[0])  
style = {**"vertex\_size"**: v\_size, **"vertex\_shape"**: **'circle'**, **"vertex\_color"**: dye,  
 **"vertex\_label\_size"**: label\_size, **"edge\_width"**: edge\_width,  
 **"layout"**: my\_lay, **"bbox"**: (600, 500), **"margin"**: 25}  
*# ###########################*ig.plot(s, **'prim\_tree.png'**, \*\*style) *# 决策树可视化及保存*

1. **实验结果**

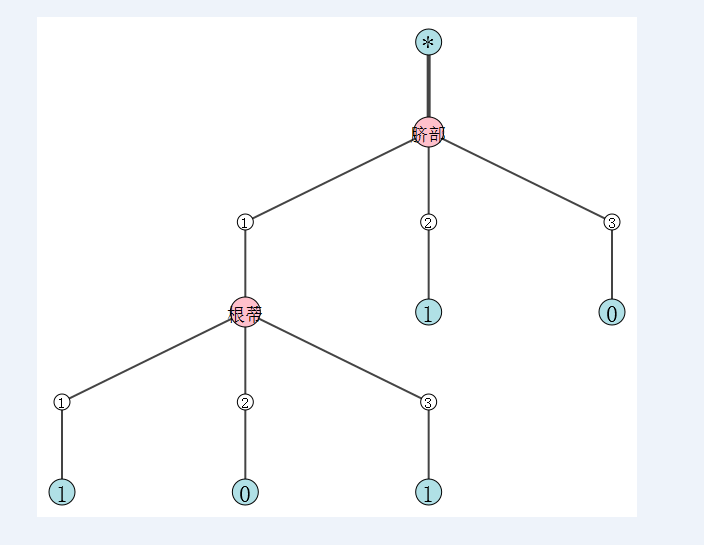
未剪枝的字典与图如下：

{'脐部1': {1: {'根蒂1': {1: 1, 2: 0, 3: 1}}, 2: {'根蒂1': {1: 0, 2: {'色泽1': {1: 1, 2: {'纹理1': {1: 0, 2: 1, 3: 1}}, 3: 1}}, 3: 1}}, 3: 0}}



预剪枝的字典与图如下：

{'脐部1': {1: {'根蒂1': {1: 1, 2: 0, 3: 1}}, 2: 1, 3: 0}}



后剪枝的字典与图如下：

{'脐部1': {1: {'根蒂1': {1: 1, 2: 0, 3: 1}}, 2: {'根蒂1': {1: 0, 2: {'色泽1': {1: 1, 2: 1, 3: 1}}, 3: 1}}, 3: 0}}

