中山大学数据科学与计算机学院

计算机科学与技术专业-人工智能

本科生实验报告

（2018-2019学年秋季学期）

课程名称：**Artificial Intelligence**

|  |  |  |  |
| --- | --- | --- | --- |
| 教学班级 | 计科2班 | 专业（方向） | 计算机科学与技术 |
| 学号 | 16337341 | 姓名 | 朱志儒 |

## 实验题目

决策树

## 实验内容

* **算法原理**

**决策树**

决策树是一种树形结构，可以是二叉树或非二叉树，树中每个非叶节点表示一个特征属性上的测试，每个分支代表这个特征属性在某个值域上的输出，每个叶节点存放一个类别。使用决策树进行分类的过程就是从根节点开始，测试待分类项中相应的特征属性，按照其值选择输出分支，直到到达叶子节点，将叶子节点存放的类别作为分类结果。

**ID3**

ID3模型是以信息熵和信息增益作为衡量标准的分类模型。

熵是指信息的混乱程度，熵值越大，变量的不确定性也就越大，计算信息熵的公式：

其中，，为类别在样本中出现的概率。

条件熵是指在已知第二个随机变量X的值的前提下，随机变量Y的信息熵。计算特征A对数据集S的条件熵的公式：

其中，A表示样本特征，是特征A所有的取值集合，V是A中一个特征值，是S中A的值为V的样例集合。

信息增益是指在某个条件下，信息复杂度，即不确定性，减少的程度。计算信息增益的公式：

其中，A表示样本特征。

在构建决策树时，选择信息增益最大的特征作为决策点。

**C4.5**

C4.5模型在ID3模型的基础上稍作改进，C4.5是以信息增益率作为衡量标准的分类模型。C4.5克服了ID3的一个缺点：用信息增益选择特征时偏向于选择分枝比较多的特征值；

计算特征A对数据集的信息增益的公式：

计算数据集D关于特征A的值的熵的公式：

这个值表示通过将数据集D划分成对应于特征A测试的v个输出的v个划分产生的信息。

计算信息增益率的公式：

在构建决策树时，选择信息增益率最大的特征作为决策点。

**CART**

CART模型生成的决策树是二叉树，CART是以GINI指数作为衡量标准的分类模型。

GINI指数是一种不等性度量，通常用来度量收入不平衡，也可用来度量任何不均匀分布，与熵的概念相似，总体内包含的类别越多，GINI指数就越大。GINI指数为介于0~1之间的数，0表示完全相等，1表示完全不相等。

对于一个数据集D包含来自n个类的样本，计算GINI指数的公式：

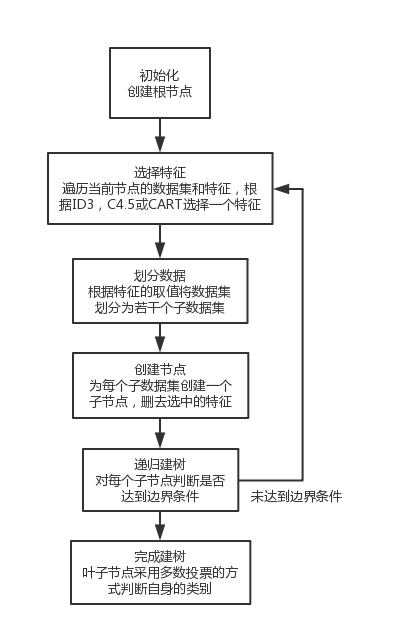
其中，是类j在D中的相对频率。

如果一个数据集D被分成两个子集D1和D2大小分别为N1和N2 , 数据包含来自n个类的样本, 则计算GINI指数的公式：

选择具有最小的属性作为分裂节点的属性，对每个属性均需要遍历所有可能的分裂位置点。

* **流程图&伪代码**

**构建决策树的流程图：**



* **关键代码**

1. **计算数据集D的信息熵：**

def empirical\_entropy(train\_set, D):

'''根据数据集train\_set计算类别D的信息熵'''

dict\_of\_kinds = {}

for i in range(len(train\_set)):

label = train\_set[i][D]

if label not in dict\_of\_kinds.keys():

dict\_of\_kinds[label] = 1

else:

dict\_of\_kinds[label] += 1

summ = len(train\_set)

for key in dict\_of\_kinds.keys():

pd = dict\_of\_kinds[key] / summ

dict\_of\_kinds[key] = pd \* math.log(pd)

return -sum(list(dict\_of\_kinds.values()))

1. **计算特征A对数据集D的条件熵：**

def condition\_entropy(train\_set, A, D):

'''根据数据集train\_set，在已知条件A的前提下，计算D的条件熵'''

dict\_of\_kinds = {}

for i in range(len(train\_set)):

label = train\_set[i][A]

if label not in dict\_of\_kinds.keys():

dict\_of\_kinds[label] = [i]

else:

dict\_of\_kinds[label].append(i)

for key in dict\_of\_kinds.keys():

dict\_of\_acct = {}

for j in dict\_of\_kinds[key]:

label = train\_set[j][D]

if label not in dict\_of\_acct.keys():

dict\_of\_acct[label] = 1

else:

dict\_of\_acct[label] += 1

summ = len(dict\_of\_kinds[key])

for keyy in dict\_of\_acct.keys():

pd = dict\_of\_acct[keyy] / summ

dict\_of\_acct[keyy] = pd \* math.log(pd)

dict\_of\_kinds[key] = len(dict\_of\_kinds[key]) / len(train\_set) \* (-sum(list(dict\_of\_acct.values())))

return sum(list(dict\_of\_kinds.values()))

1. **计算信息增益：**

def informatin\_gain(train\_set, D, A):

'''计算信息增益'''

return empirical\_entropy(train\_set, D) - condition\_entropy(train\_set, A, D)

1. **计算信息增益率：**

def information\_gain\_ratio(train\_set, D, A):

'''计算信息增益率'''

return informatin\_gain(train\_set, D, A) / empirical\_entropy(train\_set, A)

1. **计算GINI指数：**

def gini\_index(train\_set, D, A, set):

'''计算特征A的条件下，标签D的GINI指数'''

dict\_of\_kinds = {0: [], 1: []}

for i in range(len(train\_set)):

if train\_set[i][A] in set:

dict\_of\_kinds[0].append(i)

else:

dict\_of\_kinds[1].append(i)

for key in dict\_of\_kinds.keys():

dict\_of\_acct = {}

for j in dict\_of\_kinds[key]:

label = train\_set[j][D]

if label not in dict\_of\_acct.keys():

dict\_of\_acct[label] = 1

else:

dict\_of\_acct[label] += 1

summ = len(dict\_of\_kinds[key])

for kk in dict\_of\_acct.keys():

pd = dict\_of\_acct[kk] / summ

dict\_of\_acct[kk] = pd \*\* 2

dict\_of\_kinds[key] = len(dict\_of\_kinds[key]) / len(train\_set) \* (1 - sum(list(dict\_of\_acct.values())))

return sum(list(dict\_of\_kinds.values()))

1. **定义节点类**

class decision\_node:

'''定义节点类'''

def \_\_init\_\_(self, col, value=None, child\_node=None):

self.col = col

self.value = value

self.child\_node = child\_node

1. **构建决策树**

def build\_tree(self, remain\_set, used\_col, function):

'''构建决策树'''

if self.is\_same(remain\_set):

'''数据集D中的样本属于同一类别C，则将当前结点标记为C类叶结点'''

return decision\_node(-1, value=remain\_set[0][self.D])

if len(used\_col) == len(self.dict\_of\_labels.keys()):

'''特征集A为空集，或数据集D中所有样本在A中所有特征上取值相同，则将当前结点标记为叶结点，类别为D中出现最多的类'''

return decision\_node(-1, value=self.find\_mode(remain\_set))

if function == gini\_index:

'''构建CART模型决策树'''

gini = {}

gini\_num = []

for i in self.labels:

if i not in used\_col:

child\_set = get\_child\_set(self.dict\_of\_labels[i])

child\_gini = []

for set in child\_set:

child\_gini.append(gini\_index(remain\_set, self.D, i, set))

minn = min(child\_gini)

index = child\_gini.index(minn)

gini[minn] = child\_set[index]

gini\_num.append(minn)

else:

gini[9999] = []

gini\_num.append(9999)

'''选择GINI指数最小的特征作为决策点'''

choose = gini\_num.index(min(gini\_num))

left\_set = gini[min(gini\_num)]

right\_set = []

for value in self.dict\_of\_labels[choose]:

if value not in left\_set:

right\_set.append(value)

set = [left\_set, right\_set]

new\_used\_col = used\_col + [choose]

child\_node = {}

for i in range(len(set)):

new\_remain\_set = []

for row in remain\_set:

if row[choose] in set[i]:

new\_remain\_set.append(row)

strings = ''

for item in set[i]:

strings += item + '|'

if len(new\_remain\_set) == 0:

'''数据集D为空集，则将当前结点标记为叶结点，类别为父结点中出现最多的类'''

child\_node[strings] = decision\_node(-1, value=self.find\_mode(remain\_set))

else:

child\_node[strings] = self.build\_tree(new\_remain\_set, new\_used\_col, function)

else:

'''构建ID3或C4.5模型决策树'''

entropies = []

for i in self.labels:

if i not in used\_col:

entropies.append(function(remain\_set, self.D, i))

else:

entropies.append(-1)

'''选择信息增益或信息增益率最大的特征作为决策点'''

choose = entropies.index(max(entropies))

new\_used\_col = used\_col + [choose]

child\_node = {}

for label in self.dict\_of\_labels[choose]:

new\_remain\_set = []

for row in remain\_set:

if row[choose] == label:

new\_remain\_set.append(row)

if len(new\_remain\_set) == 0:

'''数据集D为空集，则将当前结点标记为叶结点，类别为父结点中出现最多的类'''

child\_node[label] = decision\_node(-1, value=self.find\_mode(remain\_set))

else:

child\_node[label] = self.build\_tree(new\_remain\_set, new\_used\_col, function)

return decision\_node(choose, child\_node=child\_node)

1. **根据生成的决策树分类数据**

def classify(self, test\_set):

'''根据已生成的决策树将test\_set中的数据分类'''

results = []

for row in test\_set:

head = self.decision\_tree

while head.col != -1:

if self.function != gini\_index:

head = head.child\_node[row[head.col]]

else:

for key in head.child\_node.keys():

if key.find(row[head.col]) != -1:

head = head.child\_node[key]

break

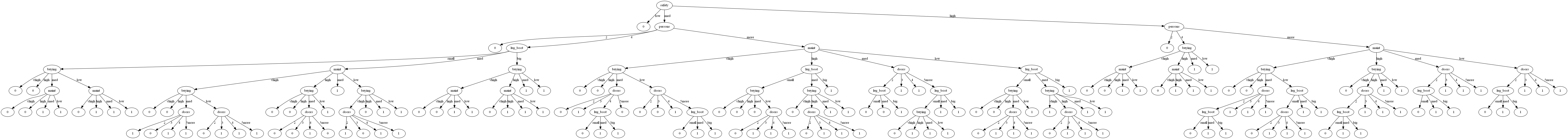
results.append(head.value)

return results

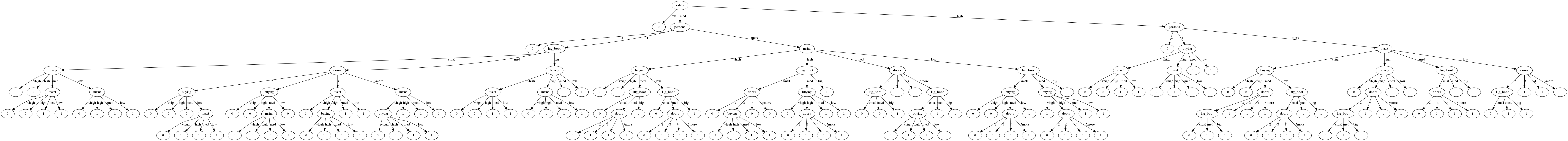
## 实验结果及分析

* **实验结果展示**

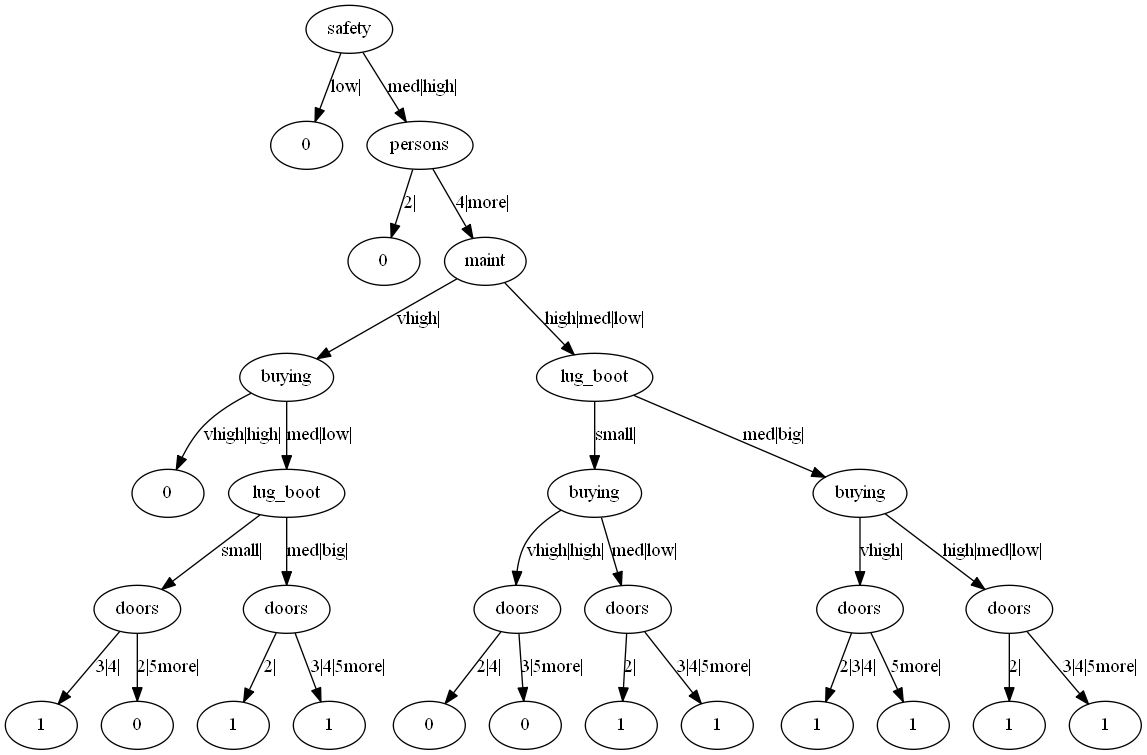
ID3模型决策树如图所示（高清图见文件ID3\_Decision\_Tree.gv.png）：



C4.5模型决策树如图所示（高清图见文件C4.5\_Decision\_Tree.gv.png）：



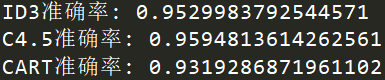
CART模型决策树如图所示（高清图见文件CART\_Decision\_Tree.gv.png）：



注：ID3\_Decision\_Tree.gv.png、C4.5\_Decision\_Tree.gv.png、CART\_Decision\_Tree.gv.png三个图片文件均可放大

* **评测指标展示**

本次试验将Car\_train.csv中的数据分为两类，前1000条数据作为训练集，其余部分作为验证集，三种模型的准确率如下：



由此可以看出使用C4.5模型效果最佳，准确率高达95.9%。

## 思考题

* **决策树有哪些避免过拟合的方法？**

避免决策树过拟合的方法：获取更多的数据、预剪枝、后剪枝、生成随机森林。

* **C4.5相比于ID3的优点是什么，C4.5又可能有什么缺点？**

C4.5相比于ID3的优点：

1. 用信息增益比来选择属性，克服了用信息增益选择属性时偏向选择取值多的属性的不足；
2. 能够完成对连续属性的离散化处理；
3. 能够对不完整数据进行处理。

C4.5的缺点：

在构造树的过程中，需要对数据集进行多次的顺序扫描和排序，因而导致算法的低效。

* **如何用决策树来进行特征选择（判断特征的重要性）？**