**云南大学软件学院**

**实 验 报 告**

姓名：王子陈 学号：20171050008 专业:电子科学与技术 日期：2019/10/23 成绩：   
任课教师：谢仲文

**数据挖掘技术 实验二**

1. **实验目的**
2. 掌握决策树分类的ID3算法、C4.5算法和CART算法。
3. 选择一种编程语言实现C4.5算法。
4. **实验内容**

1．在一个简单的、虚拟的数据集（训练集）上应用C4.5算法。该数据集如下：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 天气 | 温度 | 湿度 | 刮风 | 适合运动 |
| 1 | sunny | 85 | 85 | FALSE | no |
| 2 | sunny | 80 | 90 | TRUE | no |
| 3 | overcast | 83 | 86 | FALSE | yes |
| 4 | rainy | 70 | 96 | FALSE | yes |
| 5 | rainy | 68 | 80 | FALSE | yes |
| 6 | rainy | 65 | 70 | TRUE | no |
| 7 | overcast | 64 | 65 | TRUE | yes |
| 8 | sunny | 72 | 95 | FALSE | no |
| 9 | sunny | 69 | 70 | FALSE | yes |
| 10 | rainy | 75 | 80 | FALSE | yes |
| 11 | sunny | 75 | 70 | TRUE | yes |
| 12 | overcast | 72 | 90 | TRUE | yes |
| 13 | overcast | 81 | 75 | FALSE | yes |
| 14 | rainy | 71 | 91 | TRUE | no |

2．构建预测数据集，并在数据集上应用分类模型。

3. 简要对比ID3算法、C4.5算法和CART算法。

**三、****实验要求**

1. 完成实验内容，源码作为实验报告附件一起打为一个压缩包提供。该压缩包要包含实验报告、代码文件。
2. 关键部分要求有注释，注释量不低于20%
3. 要求独立完成，不得抄袭代码。

**四、关键实验步骤（请粘贴关键步骤、代码、实验结果）**

**(使用python语言)**

**#计算信息熵**

def cal\_Ent(data\_set):

num=len(data\_set) #记录条数

label\_counts={} #字典放类别(yes/no)

for i in range(num): #遍历data\_set中的每一条记录

current\_label=data\_set[i][-1] #只取最后一列数据(Play?)

if current\_label not in label\_counts.keys(): #如果这个属性没在label\_counts中出现

label\_counts[current\_label]=0 #就把这个键的值赋为0

label\_counts[current\_label]+=1 #让当前键的值+1

Ent=0.0

for key in label\_counts:

prob=float(label\_counts[key])/num

Ent=Ent-prob\*log(prob,2)

return Ent

# In[2]:

**#离散属性,切割数据集,返回不包含axis列的子数据集**

def split\_data\_set(data\_set, axis, value):

ret\_data\_set = []

for feat\_vec in data\_set:

if feat\_vec[axis] == value:

reduce\_feat\_vec = feat\_vec[:axis] #去掉axis那个值

reduce\_feat\_vec.extend(feat\_vec[axis + 1:])

ret\_data\_set.append(reduce\_feat\_vec)

return ret\_data\_set

#split\_data\_set(data\_set1,0,"rainy")

# In[3]:

**#按连续属性切割数据集的方法:**

def split\_con\_data\_set(data\_set,axis,value,LorR='L'):

return\_data\_set=[]

if LorR =='L': #构建左子树

for feature\_vec in data\_set:

if feature\_vec[axis]<=value: #属性值小于等于value分在左子树

reduce\_feat\_vec = feature\_vec[:axis]

reduce\_feat\_vec.extend(feature\_vec[axis + 1:])

return\_data\_set.append(reduce\_feat\_vec)

else: #大于value分在右子树

for feature\_vec in data\_set:

if feature\_vec[axis]>value:

reduce\_feat\_vec = feature\_vec[:axis]

reduce\_feat\_vec.extend(feature\_vec[axis + 1:])

return\_data\_set.append(reduce\_feat\_vec)

return return\_data\_set

# In[4]:

**#把连续属性离散化,排序以后,只有在类别发生改变的地方才需要切开,找到使信息增益率最大的分类方式**

best\_division=[0]#用这个列表存分界线,初始值为0

def cal\_continue\_gain\_ratio (data\_set\_raw,attribute):

data\_set=sorted(data\_set\_raw,key=operator.itemgetter(labels.index(attribute)),reverse=False)#先排序

num=len(data\_set)

min\_entropy=cal\_Ent(data\_set) #初始为未分类的信息熵

left\_sub\_data\_set=[]

right\_sub\_data\_set=[]

for i in range(num-1):#从第一条记录遍历到倒数第二条,因为下面有i+1

if data\_set[i][-1]!= data\_set[i+1][-1]:#如果上下两条的类别不同,就划分一次

for j in range(num):

if j<=i: #把在分界线之上的记录,分在左子树

left\_sub\_data\_set.append(data\_set[j])

else: #在分界线之下的就分在右子树

right\_sub\_data\_set.append(data\_set[j])

#对这次的分类结果计算条件熵

prob\_L=len(left\_sub\_data\_set)/num

prob\_R=len(right\_sub\_data\_set)/num

sigma\_ent=(prob\_L)\*cal\_Ent(left\_sub\_data\_set)+(prob\_R)\*cal\_Ent(right\_sub\_data\_set)

if sigma\_ent<min\_entropy:

min\_entropy=sigma\_ent

best\_division[0]=i #最佳分界线

left\_sub\_data\_set.clear()

right\_sub\_data\_set.clear()

max\_gain=cal\_Ent(data\_set)-min\_entropy #计算最大信息增益

#计算分类信息度量H

H=0.0

prob\_l=(best\_division[0]+1)/num

prob\_r=1-prob\_l

H=-prob\_l\*log(prob\_l,2)-prob\_r\*log(prob\_r,2)

#计算信息增益率

gain\_ratio=max\_gain/H

return gain\_ratio

# In[5]:

**#计算离散属性的gain\_ratio(天气/刮风)**

def cal\_discrete\_gain\_ratio(data\_set,attribute):

base\_entropy=cal\_Ent(data\_set) #未分类时的信息熵

info\_gain=0 #信息增益初始为0

num\_records=len(data\_set) #父节点的记录条数

feature\_val\_list=[]#用来存储每一条记录的第label\_idx个属性的特征值

for i in range(num\_records):#遍历每一条记录

feature\_val\_list.append(data\_set[i][labels.index(attribute)])

unique\_feature\_val\_list=set(feature\_val\_list) #得到所有的离散特征

#计算不同类特征的信息熵,再乘上权重求和就是条件熵

sigma=0

H=0

for feature\_val in unique\_feature\_val\_list: #遍历每一种特征

sub\_data\_set=split\_data\_set(data\_set,labels.index(attribute),feature\_val) #去掉该特征值,得到以该特征分类的子树

prob=len(sub\_data\_set)/num\_records

sigma+=prob\*cal\_Ent(sub\_data\_set)

#计算分裂度量信息度量H

H=H-prob\*log(prob,2)

info\_gain=base\_entropy-sigma

#计算信息增益率

gain\_ratio=info\_gain/H

return gain\_ratio

# In[6]:

**#找到最能有效分类的属性**

def choose\_best\_attribute\_to\_split(data\_set,labels):

if len(labels)==1: #如果只剩1个属性了,最佳属性就是它

return labels[0]

best\_gain\_ratio=0

best\_attribute=labels[-1]

for attribute in labels:

if attribute=='weather' or attribute=='windy':

gain\_ratio=cal\_discrete\_gain\_ratio(data\_set,attribute)

else:

gain\_ratio=cal\_continue\_gain\_ratio(data\_set,attribute)

if gain\_ratio>best\_gain\_ratio:

best\_gain\_ratio=gain\_ratio

best\_attribute=attribute

return best\_attribute

# In[7]:

**#统计yes和no出现的次数,并按大小排序,返回出现次数多的类别**

def majority\_count(class\_list):

class\_count={} #类别统计

for vote in class\_list: #yes和no两种

if vote not in class\_count.keys():

class\_count[vote]=0

class\_count[vote]+=1

sorted\_class\_count=sorted(class\_count.items(),key=operator.itemgetter(1))

return sorted\_class\_count[-1][0]

# In[8]:

**#构建决策树**

def create\_tree(data\_set,labels):

class\_list=[records[-1] for records in data\_set] #只取类别:yes/no

if class\_list.count(class\_list[0])==len(class\_list):#最后一个元素的个数等于列表长度,说明很"纯",里面只  
 有一种类别

return class\_list[0] #就可以返回分类结果:yes / no

if len(data\_set[0])==1: #如果所有属性都被遍历完了,

return majority\_count(class\_list) #返回出现次数最多的类别

#按照信息增益率最大选取分类特征属性

best\_attribute=choose\_best\_attribute\_to\_split(data\_set,labels)#返回最优划分属性

#以最优属性开始划分:

#如果是离散属性(天气和刮风),庶出的每一颗子树不包含最优属性:

if best\_attribute=='weather'or best\_attribute=='windy':

best\_attribute\_label=best\_attribute #字典的key值

decision\_tree={best\_attribute\_label:{}} #构建树的字典

feature\_values=[example[labels.index(best\_attribute)]for example in data\_set]#把最优属性的一列取出

unique\_feature\_values=set(feature\_values) #取出互不重复的特征值集合

#print(unique\_feature\_values)

axis=labels.index(best\_attribute)

del(labels[labels.index(best\_attribute)]) #从labels[ ]中删除该label,已经选择过的属性不再参与分类

sub\_labels=labels[:] #浅拷贝

for feature\_value in unique\_feature\_values:#遍历每一种特征

sub\_data\_set=split\_data\_set(data\_set,axis,feature\_value)#得到每个特征划分的子树

decision\_tree[best\_attribute\_label][feature\_value]=create\_tree(sub\_data\_set,sub\_labels)#每个节点下面再分树

sub\_labels=labels[:]

#如果是连续属性,,分别构建左子树和右子树

else:

best\_attribute\_label=best\_attribute+'<='+str(data\_set[best\_division[0]][labels.index(best\_attribute)])

decision\_tree={best\_attribute\_label:{}} #构建树的字典

axis=labels.index(best\_attribute) #找到划分轴

left\_sub\_data\_set=split\_con\_data\_set(data\_set,axis,data\_set[best\_division[0]][labels.index(best\_attribute)],'L') right\_sub\_data\_set=split\_con\_data\_set(data\_set,axis,data\_set[best\_division[0]][labels.index(best\_attribute)],'R')

del(labels[axis])

sub\_labels=labels[:]

decision\_tree[best\_attribute\_label]['yes']=create\_tree(left\_sub\_data\_set,sub\_labels)

decision\_tree[best\_attribute\_label]['no']=create\_tree(right\_sub\_data\_set,sub\_labels)

return decision\_tree

代入数据集:

data\_set=[['overcast',64,65,'TRUE','yes'],['rainy',65,70,'TRUE','no'],

['rainy',68,80,'FALSE','yes'],['sunny',69,70,'FALSE','yes'],

['rainy',70,96,'FALSE','yes'],['rainy',71,91,'TRUE','no'],

['sunny',72,95,'FALSE','no'],['overcast',72,90,'TRUE','yes'],

['rainy',75,80,'FALSE','yes'],['sunny',75,70,'TRUE','yes'],

['sunny',80,90,'TRUE','no'],['overcast',81,75,'FALSE','yes'],

['overcast',83,86,'FALSE','yes'],['sunny',85,85,'FALSE','no']]

labels=['weather','temperature','humidity','windy']

mytree=create\_tree(data\_set,labels)

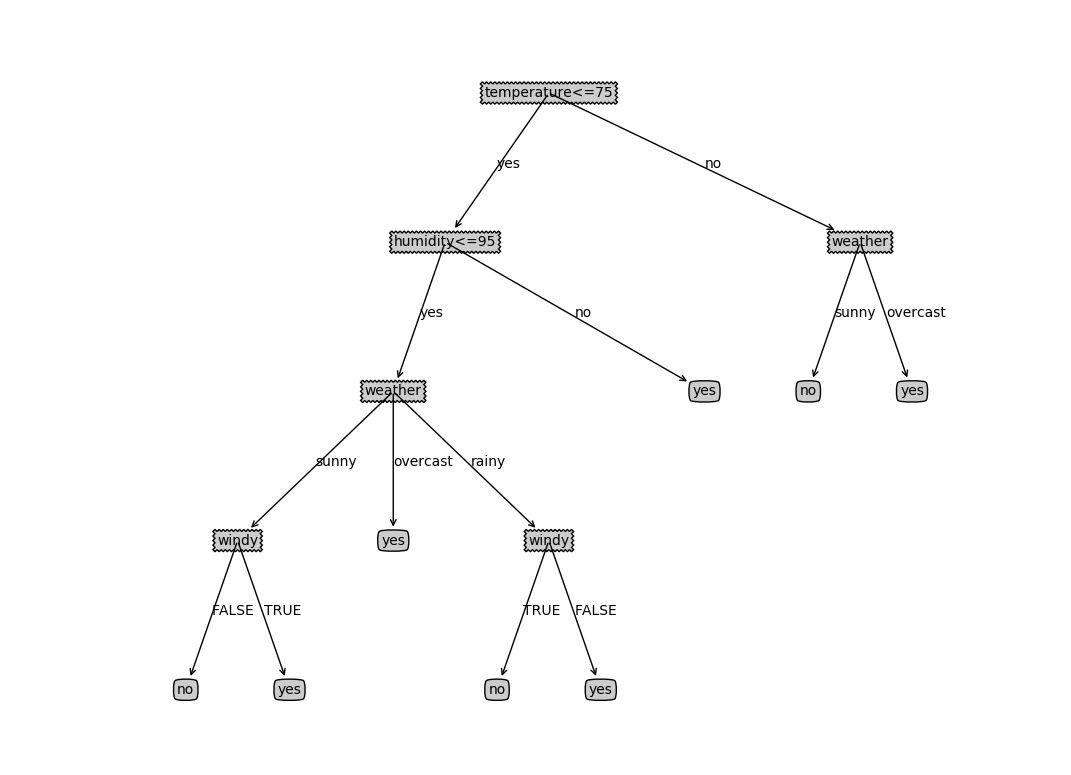
print(mytree)

create\_plot(mytree)

运行结果:是一个字典

{'temperature<=75': {'yes': {'humidity<=95': {'yes': {'weather': {'sunny': {'windy': {'FALSE': 'no', 'TRUE': 'yes'}}, 'rainy': {'windy': {'TRUE': 'no', 'FALSE': 'yes'}}, 'overcast': 'yes'}}, 'no': 'yes'}}, 'no': {'weather': {'sunny': 'no', 'overcast': 'yes'}}}}

字典形式的决策树仍然不易理解，利用Matplotlib库的annotate（注释）模块绘制决策树，就可以很直观的看出决策树的结构。



**测试决策树**:  
用原数据集重新放入决策树,进行分类,得到的类别与实际值对比:(有一条记录不同)

由此看出,得到的决策树的准确率为71.4%

测试分类部分的代码:  
def classify(inputTree,featLabels,testVec):

# 得到树中的第一个特征

firstStr = list(inputTree.keys())[0]

# 得到第一个对应的值

secondDict = inputTree[firstStr]

if firstStr.find('<')==-1:

firstLabel=firstStr

else:

firstLabel=firstStr[:firstStr.find('<')]

# 得到树中第一个特征对应的索引

featIndex = featLabels.index(firstLabel)

classLabel=None

# 遍历树

for key in secondDict.keys():

if featLabels[featIndex]=='weather'or featLabels[featIndex]=='windy':

# 如果在secondDict[key]中找到了testVec[featIndex]

if testVec[featIndex] == key:

# 判断secondDict[key]是否为字典

if type(secondDict[key]).\_\_name\_\_ == 'dict':

# 若为字典，递归的寻找testVec

classLabel = classify(secondDict[key], featLabels, testVec)

else:

# 若secondDict[key]为标签值，则将secondDict[key]赋给classLabel

classLabel = secondDict[key]

else: #连续属性：温度和湿度

splitValue = eval(firstStr[firstStr.find('=')+1:])

if testVec[featIndex]<=splitValue: #进入左子树

if type(secondDict['yes']).\_\_name\_\_=='dict': #该分支不是叶子节点,递归

classLabel=classify(secondDict['yes'],featLabels,testVec)

else: #如果是叶子,返回结果

classLabel=secondDict['yes']

else:

if type(secondDict['no']).\_\_name\_\_=='dict':

classLabel=classify(secondDict['no'],featLabels,testVec)

else:#如果是叶子节点,返回结果

classLabel=secondDict['no']

# 返回类标签

return classLabel

**简要对比ID3算法、C4.5算法和CART算法。**

ID3算法是在决策树各个节点上应用信息增益准则选择特征递归地构建决策树。信息增益大, 说明使用该特征后划分得到的子集纯度越高，即不确定性越小。**缺点：**信息增益偏向取值较多的特征（原因：当特征的取值较多时，根据此特征划分更容易得到纯度更高的子集，因此划分后的熵更低，即不确定性更低，因此信息增益更大）

C4.5算法是对ID3算法做了改进，在生成决策树过程中采用信息增益比来选择特征。信息增益会偏向取值较多的特征，使用信息增益比可以对这一问题进行校正。能够完成对连续属性的离散化处理；能够对不完整数据进行处理。

基尼指数Gini(D)表示集合D的不确定性，基尼指数Gini(D,A)表示经A分割后集合D的不确定性。基尼指数值越大，样本集合的不确定性也就越大, 选Gini指数最小的特征作为分裂标准。CART和C4.5支持数据特征为连续分布时的处理，主要通过使用二元切分来处理连续型变量，即求一个特定的值——分裂值：特征值小于等于分裂值就走左子树，或者就走右子树。