北京邮电大学软件学院

2019-2020 学年第一学期实验报告

课程名称： 大作业

项目名称： 决策树算法

项目完成人：

姓名： 司洪超 **学号：** 2019180154

姓名： 王磊 **学号：** 2019180163

姓名： _____ **学号：** _____

姓名： _____ **学号：** _____

姓名： _____ **学号：** _____

指导教师： 李朝晖

日　　期： 2019 年 12 月 10 日

一、 实验目的

1、 深刻理解并掌握决策树算法思想；
2、 提高应用决策树算法求解问题的技能；

二、 实验内容

1、代码实现决策树算法
2、理解熵和条件熵、ID3 和 C4.5 算法的思想

三、 实验环境

Python3.7 环境

四、 实验结果

熵（entropy）

在信息论与概率论中，熵（entropy）用于表示"随机变量不确定性的度量"

所以下面我们说一下信息量熵的定义。

设 X 是一个有限状态的离散型随机变量，其概率分布为：

$$P(X = x_i) = p_i,\ i = 1, 2, \cdots, n$$

则随机变量 X 的熵定义为：

$$H(X) = -\sum_{i=1}^{n} p_i log(p_i)$$

熵越大，则随机变量的不确定性越大。

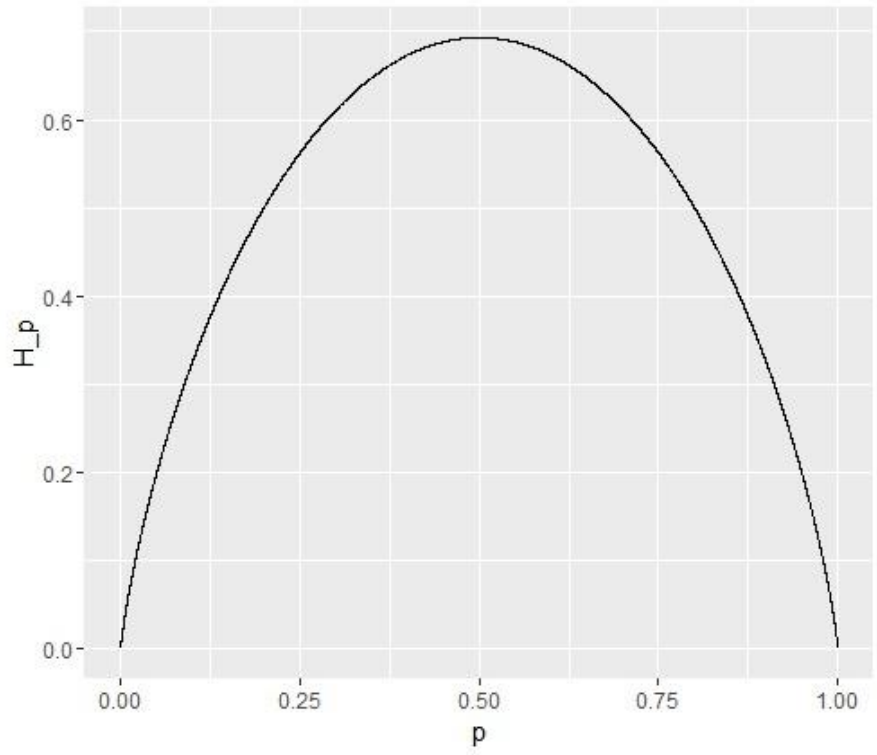
当随机变量只有 0 和 1 两种取值的时候，假设 P（X=1）=p，则有：

$$H(X) = -plog(p) - (1-p)log(1-p)$$

从而有：

$$\frac{\partial H(X)}{\partial p} = -log(\frac{p}{1-p})$$

从而可知，当 p=0.5 时，熵取值最大，随机变量不确定性最大。如图：



条件熵（conditional entropy）

随机变量 X 给定的条件下，Y 的条件概率分布的熵对 X 的数学期望，其数学推导如下：

$$
\begin{aligned}
H(Y/X) &= \sum_{x \in X} P(x) H(Y/X=x) \\
&= -\sum_{x \in X} P(x) \sum_{y \in Y} P(y/x) \log P(y/x) \\
&= -\sum_{x \in X} \sum_{y \in Y} P(x,y) \log P(y/x)
\end{aligned}
$$

随机变量 Y 的条件熵 H（Y|X）定义为：

$$H(Y|X) = \sum_{i=1}^{n} p_i H(Y|X=x_i)$$

其中：

$$p_i = P(X=x_i)$$

条件熵 H(Y|X)表示在已知随机变量 X 的条件下随机变量 Y 的不确定性。注意一下，条件熵中 X 也是一个变量，意思是在一个变量 X 的条件下（变量 X 的每个值都会取到），另一个变量 Y 的熵对 X 的期望。

信息增益（information gain）

信息增益表示的是：得知特征 X 的信息而使得类 Y 的信息的不确定性减少的程度。简单说，就是当我们用另一个变量 X 对原变量 Y 分类后，原变量 Y 的不确定性就会减少了（即熵值减少）。而熵就是不确定性，不确定程度减少了多少其实就是信息增益，这就是信息增益的由来。

所以信息增益的具体定义如下：

特征 A 对训练数据集 D 的信息增益 g(D，A）定义为集合 D 的经验熵 H（D）与特征 A 给定条件下 D 的经验条件熵 H（D/A）之差，即：

$$g(D,A) = H(D) - H(D|A)$$

一般地，熵 H(Y)与条件熵 H(Y|X)之差成为互信息（mutual information）。

根据信息增益准则而进行特征选择的方法是：对训练数据集 D，计算其每个特征的信息增益，并比他们大小，从而选择信息增益最大的特征。

假设训练数据集为 D，样本容量为|D|，由 k 个类别 C_k，$|C_k|$为类别 C_k 的样本个数，某一特征 A 由 n 个不同的取值 a1,a2,a3,.....an。根据特征 A 的取值可将数据集 D 划分为 n 个子集 D1，D2.....D_n，$|D_i|$为 D_i 的样本个数，并记子集 Di 中属于类 C_k，的样本的集合为 D_{ik}，$|D_{ik}|$为 D_{ik} 的样本个数。

则信息增益的算法如下：

- -输入：训练数据集 D 和特征 A
- -输出：特征 A 对训练数据集 D 的信息增益 g(D,A)

（1） 计算数据集 D 的经验熵 H(D)。

$$H(D) = -\sum_{k=1}^{K} \frac{|C_k|}{|D|} log \frac{|C_k|}{|D|}$$

（2）计算特征 A 对数据集 D 的经验条件 H（D|A）

$$H(D|A) = \sum_{i=1}^{n} \frac{|D_i|}{|D|} H(D_i) = \sum_{i=1}^{n} \frac{|D_i|}{|D|} \sum_{k=1}^{K} \frac{|D_{ik}|}{D_i} log \frac{|D_{ik}|}{D_i}$$

（3）计算信息增益

$$g(D,A) = H(D) - H(D|A)$$

信息增益比（information gain ratio）

以信息增益作为特征选择准则，会存在偏向于选择取值较多的特征的问题，可以采用信息增益比对这个问题进行校正。

特征 A 对训练数据集 D 的信息增益比定义为其信息增益与训练集 D 关于特征 A 的值的熵之比，即：

$$g_R(D|A) = \frac{g(D,A)}{H_A(D)}$$

其中：

$$H_A(D) = -\sum_{i=1}^{n} \frac{|D_i|}{|D|} log \frac{|D_i|}{|D|}.$$

ID3 算法

ID3 算法所采用的度量标准就是我们前面提到的"信息增益"。当属性 a 的信息增益最大时，则意味着用 a 属性划分，其所获得的"纯度"提升最大，我们所要做的，就是找到信息增益最大的属性。

ID3 算法的核心是在决策树的各个节点上应用信息增益准则进行特征选择，具体的做法是：

- 从根节点上开始，对结点计算所有可能特征的信息增益，选择信息增益最大的特征作为结点的特征，并由该特征的不同取值构建子节点；
- 对于子节点递归的调用以上方法，构建决策树；
- 直到所有特征的信息增益均很小或者没有特征可选择的时候为止。

ID3 算法存在的缺点：

1. ID3 算法在选择根节点和内部节点中的分支属性时，采用信息增益作为评价标准。信息增益的缺点是倾向于选择取值较多是属性，在有些情况下这类属性可能不会提供太多有价值的信息。

2. ID3 算法只能对描述属性为离散型属性的数据集构造决策树 。

为了改进决策树，又提出了 ID4.5 算法和 CART 算法。

C4.5 算法

C4.5 算法与 ID3 算法的区别主要是在于它在生产决策树的过程中，使用信息增益比来进行特征选择。

实际上，信息增益准则对于可取值数目较多的属性会有所偏好，为了减少这种偏好可能带来的不利影响，C4.5 决策树算法不直接使用信息增益，而是使用"信息增益率"来选择最优划分属性，信息增益率定义为：

$$Gain\_ratio(Y,X) = \frac{Gain(Y,X)}{H(X)}$$

其中，分子为信息增益，分母为属性 X 的熵。

需要注意的是，增益率准则对可取值数目较少的属性有所偏好。

所以一般这样选取划分属性：先从候选属性中找到信息增益高于平均水平的属性，再从中选择增益率最高的。

五、 附录

实验代码

```
import matplotlib.pyplot as plt

# 定义文本框和箭头格式 （树节点格式的常量）
decisionNode = dict(boxstyle='sawtooth',fc='0.8')
leafNode = dict(boxstyle='round4',fc='0.8')
arrows_args = dict(arrowstyle='<-')

# 绘制带箭头的注解
def plotNode(nodeTxt,centerPt,parentPt,nodeType):
    createPlot.ax1.annotate(nodeTxt,xy=parentPt,
    xycoords='axes fraction',
    xytext=centerPt,textcoords='axes fraction',
    va='center',ha='center',bbox=nodeType,
    arrowprops=arrows_args)


# 在父子节点间填充文本信息
def plotMidText(cntrPt,parentPt,txtString):
    xMid = (parentPt[0] - cntrPt[0]) / 2.0 + cntrPt[0]
    yMid = (parentPt[1] - cntrPt[1]) / 2.0 + cntrPt[1]
    createPlot.ax1.text(xMid,yMid,txtString, va="center", ha="center", rotation=30)

def plotTree(myTree,parentPt,nodeTxt):
    # 求出宽和高
    numLeafs = getNumLeafs(myData)
    depth = getTreeDepth(myData)
    firstStides = list(myTree.keys())
    firstStr = firstStides[0]
    # 按照叶子结点个数划分 x 轴
    cntrPt = (plotTree.xOff + (0.1 + float(numLeafs)) /2.0/plotTree.totalW,plotTree.yOff)
    plotMidText(cntrPt,parentPt,nodeTxt)
```

```
    plotNode(firstStr,cntrPt,parentPt,decisionNode)
    secondDict = myTree[firstStr]
    # y 方向上的摆放位置 自上而下绘制，因此递减 y 值
    plotTree.yOff = plotTree.yOff - 1.0/plotTree.totalD
    for key in secondDict.keys():
        if type(secondDict[key]).__name__  == 'dict':
            plotTree(secondDict[key],cntrPt,str(key))
        else:
            plotTree.xOff = plotTree.xOff + 1.0 / plotTree.totalW  # x 方向计算结点坐标
            plotNode(secondDict[key], (plotTree.xOff, plotTree.yOff), cntrPt, leafNode)  # 绘制
            plotMidText((plotTree.xOff, plotTree.yOff), cntrPt, str(key))  # 添加文本信息
    plotTree.yOff = plotTree.yOff + 1.0 / plotTree.totalD  # 下次重新调用时恢复 y

def myTree():
    treeData = {'voice': {'Think': {'hair': {'Long': 'female', 'Short': 'male'}}, 'Thin': 'female'}}
    return treeData

# 获取叶子节点的数目
def getNumLeafs(myTree):
    # 初始化结点数
    numLeafs = 0
    firstSides = list(myTree.keys())
    # 找到输入的第一个元素，第一个关键词为划分数据集类别的标签
    firstStr = firstSides[0]
    secondDect = myTree[firstStr]
    for key in secondDect.keys():
        # 测试节点的数据类型是否为字典
        if type(secondDect[key]).__name__ == 'dict':
            numLeafs += getNumLeafs(secondDect[key])
        else:
            numLeafs +=1
    return numLeafs

# 获取树的层数
def getTreeDepth(myTree):
    maxDepth = 0
    firstSides = list(myTree.keys())
    firstStr = firstSides[0]
    secondDict = myTree[firstStr]
    for key in secondDict.keys():
        # 测试节点的数据类型是否为字典
        if type(secondDict[key]).__name__ == 'dict':
            thisDepth = 1 + getTreeDepth(secondDict[key])
        else:
```

```
            thisDepth = 1
        if thisDepth > maxDepth:
            maxDepth = thisDepth

    return maxDepth

# 主函数
def createPlot(inTree):
    # 创建一个新图形并清空绘图区
    fig = plt.figure(1,facecolor='white')
    fig.clf()
    axprops = dict(xticks=[], yticks=[])
    createPlot.ax1 = plt.subplot(111, frameon=False, **axprops)   # no ticks
    # createPlot.ax1 = plt.subplot(111, frameon=False) #ticks for demo puropses
    plotTree.totalW = float(getNumLeafs(inTree))
    plotTree.totalD = float(getTreeDepth(inTree))
    plotTree.xOff = -0.5 / plotTree.totalW
    plotTree.yOff = 1.0
    plotTree(inTree, (0.5, 1.0), '')
    plt.show()


if __name__ == '__main__':
    myData   = myTree()
    myData['voice'][3] = 'maybe'
    print(myData)
    # LeafNum = getNumLeafs(myData)
    # TreeDepth = getTreeDepth(myData)
    # print(LeafNum)
    # print(TreeDepth)
    createPlot(myData)
```