Ecperiment3: Decision Tree Based on ID3 algorithm

Time: 2022/3/22
Location: Science_Building_119
Name: 易弘睿
Number: 20186103

Part I: Review

ID3算法：

ID3算法是一种决策树构建算法，其核心为“信息熵”（信息源的不确定程度）
ID3算法是以信息论为基础，以信息增益为衡量标准，从而实现对数据的归纳分类
ID3算法计算每个属性的信息增益，并选取具有最高增益的属性作为给定的测试属性

算法步骤：

1. 初始化特征集合和数据集合
2. 计算数据集合信息熵和所有特征的条件熵，选择信息增益最大的特征作为当前决策节点
3. 更新数据集合和特征集合（删除上一步使用的特征，并按照特征值来划分不同分支的数据集合）
4. 重复上述两步，若子集包含单一特征，则为分支叶子结点（叶子结点：出度为0的结点）

Part II: Introduction

对初始代码的修改主要如下：
1. 对代码按照不同部分功能进行命名；
2. 对代码进行每行注释；
3. 利用自定义plotNode函数来完成决策树画图（一次绘制的是一个箭头和一个节点）。

Part III: Annotation

1. 数据库的导入

In [1]:

```
import pandas as pd # 导入pandas库
import math         # 导入math库
import numpy as np  # 导入numpy
import matplotlib
import matplotlib.pyplot as plt
```

2. 数据的预处理

In [2]:

```
filePath = "西瓜.csv"                # 导入西瓜数据
data = pd.read_csv(filePath)         # 导入数据存入data变量
featureList = data.columns.tolist() # 获得feature名， 比如'色泽', '根蒂'
attr=data.columns.values.tolist()    # 将标签转换为numpy列表
attr_dict={}                         #用于记录每一个属性的取值
```

In [3]:

```
# 使用for循环，记录每个feature下的子feature名，比如色泽下的子feature就有'青绿', '乌黑', '浅白', 即生
for x in attr[:-1]:
    temp=np.array(data[x])
    attr_dict[x]=set(temp)
```

3. 主函数和子函数的定义

In [4]:

```
def countMost(data):

    '''
        函数功能:
            在特定情况下，直接通过好瓜与坏瓜的数量来执行分类，标记类别为好瓜与坏瓜中占比较多的

        输入:
            data: 原始西瓜数据

        输出:
            待分类西瓜的类别
    '''

    # 获得好瓜这一feature下的数据
    label = data['好瓜'].values.tolist()
    # 统计好瓜与坏瓜的数量，选择数量大的作为最终的类别
    temp = max(label, key=label.count)
    return temp
```

In [5]:

```
def is_same(data,featureList):
    '''
        函数功能:
            判断输入的瓜的种类是否全都一样(如果a,b两个瓜的种类全都一样, 那么a,b两个瓜的色泽,根蒂,敲声

        输入:
            data: 原始西瓜数据
            featureList: 原始西瓜数据中的feature种类, 包含色泽,根蒂,敲声,纹理,脐部,触感

        输出:
            一个布尔型的数据, True表示有多种, False表示仅有一种西瓜
    '''

    # 对西瓜进行分类并计数
    temp = data.groupby(featureList).count()

    # 如果分类后的length仅为1, 那么表明只有一种西瓜, 则返回False, 否则返回True
    if len(temp) != 1:
        return False
    else:
        return True
```

In [6]:

```
def findBestFeature(data,featureList):

    '''
        函数功能:
            判断最适合用于西瓜分类的feature

        输入:
            data：原始西瓜数据
            featureList：原始西瓜数据中的feature种类，包含色泽,根蒂,敲声,纹理,脐部,触感

        输出:
            最适合用于西瓜分类的feature
    '''

    # 计算好瓜与坏瓜的百分比
    a = data[featureList[-1]].value_counts()/len(data)

    # 计算好瓜与坏瓜的信息熵
    Ent = 0
    for i in a:
        Ent += -i*math.log(i,2)

    # 初始化不同feature的信息增益
    res = {}

    #遍历各个属性并求出相应的熵增
    for i in featureList[:-1]:
        # 获得feature_i的子feature名称
        subFeature = set(data[i].values.tolist())
        # 初始化feature_i的信息熵
        tempres = 0
        for j in subFeature:
            # 找到子feature出现的位置
            subdata = data.loc[data[i] == j]
            # 统计子feature出现的概率
            NumD = len(subdata)/len(data)
            # 计算子feature中好瓜与坏瓜的百分比
            temp = subdata[featureList[-1]].value_counts()/len(subdata)
            # 计算子feature下的好瓜与坏瓜的信息熵
            EntD = 0
            for k in temp:
                EntD += -k*math.log(k,2)
            # 根据子feature出现的概率，累加信息熵
            tempres += NumD*EntD
        # 获得feature_i的信息增益
        res[i] = Ent - tempres

    # 选取信息增益最大的作为西瓜分类的feature
    bestFeature = max(res, key=res.get)
    return bestFeature
```

In [7]:

```
def TreeGenerate(data,featureList,attr_dict):
    '''
        【主函数】
        输入:
            data: 原始西瓜数据
            featureList: 原始西瓜数据的feature种类, 包含色泽,根蒂,敲声,纹理,脐部,触感
            attr_dict: 原始西瓜数据feature下的子feature种类
        输出:
            Tree
    '''

    # 获得西瓜的好坏种类
    label = data['好瓜'].values.tolist()
    # 如果所有的西瓜都是好瓜或者都是坏瓜, 那么待分类的西瓜将会直接被标记为好瓜或者坏瓜
    if label.count(label[0]) == len(label):
        node = label[0]
        return node
    # 如果西瓜没有feature或者所有的西瓜种类完全一致, 那么将触发countMost
    elif len(featureList)==0 or is_same(data,featureList[:-1]):
        node = countMost(data)
        return node
    # 寻找最适合用于分类的feature
    bestFeature = findBestFeature(data,featureList)
    # 将最好的feature从featureList中移除
    featureList.remove(bestFeature)
    mytree = {bestFeature:{}}
    # 获得最好feature的子feature
    subFeature = attr_dict[bestFeature]
    for i in subFeature:
        # 寻找子feature_i的位置
        subdataset = data.loc[data[bestFeature]==i]
        # 如果子feature下没有西瓜, 则触发countMost机制
        if subdataset.empty:
            node = countMost(data)
            mytree[bestFeature][i] = node
        # 如果子feature下有西瓜, 则迭代, 生成新的分支节点
        else:
            feature = featureList[:]
            mytree[bestFeature][i]=TreeGenerate(subdataset,feature,attr_dict)
    return mytree
```

4. 主函数的运行

In [8]:

```
node = TreeGenerate(data,featureList,attr_dict)
```

In [9]:

```
print(node)
```

{'纹理': {'清晰': {'根蒂': {'稍蜷': {'色泽': {'青绿': '是', '浅白': '是', '乌黑': {'触感': {'硬滑': '是', '软粘': '否'}}}}, '硬挺': '否', '蜷缩': '是'}}, '稍糊': {'触感': {'硬滑': '否', '软粘': '是'}}, '模糊': '否'}}

Part IV: 利用自定义函数绘制决策树

In [10]:

```
# 对绘制是图形属性的一些定义
decisionNode = dict(boxstyle="sawtooth", fc="0.8")
leafNode = dict(boxstyle="round4", fc="0.8")
arrow_args = dict(arrowstyle="<-")
```

In [11]:

```
# 递归计算树的叶子节点个数
def getNumLeafs(myTree):
    numLeafs=0
    firstStr=list(myTree.keys())[0]
    secondDict=myTree[firstStr]
    for key in secondDict.keys():
        if type(secondDict[key]).__name__=='dict':
            numLeafs+=getNumLeafs(secondDict[key])
        else:
            numLeafs+=1
    return numLeafs
```

In [12]:

```
# 递归计算树的深度
def getTreeDepth(myTree):
    # 初始化树的深度
    maxDepth = 0
    # 获取树的第一个键名
    firstStr = list(myTree.keys())[0]
    # 获取键名所对应的值
    secondDict = myTree[firstStr]
    # 遍历树
    for key in secondDict.keys():
        # 如果获取的键是字典，树的深度加1
        if type(secondDict[key]).__name__ == 'dict':
            thisDepth = 1 + getTreeDepth(secondDict[key])
        else:
            # 去深度的最大值
            thisDepth = 1
        if thisDepth > maxDepth: maxDepth = thisDepth
    # 返回树的深度
    return maxDepth
```

In [13]:

```
# 用注释形式绘制节点和箭头线
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=arrow_args )
```

In [14]:

```
# 用来绘制线上的标注
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)
```

In [15]:

```
# 通过递归决定整个树图的绘制
def plotTree(myTree, parentPt, nodeTxt):#if the first key tells you what feat was split on
    numLeafs = getNumLeafs(myTree)
    depth = getTreeDepth(myTree)
    firstStr = list(myTree.keys())[0]
    cntrPt = (plotTree.xOff + (1.0 + float(numLeafs))/2.0/plotTree.totalW, plotTree.yOff)
    plotMidText(cntrPt, parentPt, nodeTxt)
    plotNode(firstStr, cntrPt, parentPt, decisionNode)
    secondDict = myTree[firstStr]
    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))        #recursion
        else:   #it's a leaf node print the leaf node
            plotTree.xOff = plotTree.xOff + 1.0/plotTree.totalW
            plotNode(secondDict[key], (plotTree.xOff, plotTree.yOff), cntrPt, leafNode)
            plotMidText((plotTree.xOff, plotTree.yOff), cntrPt, str(key))
    plotTree.yOff = plotTree.yOff + 1.0/plotTree.totalD
```

In [16]:

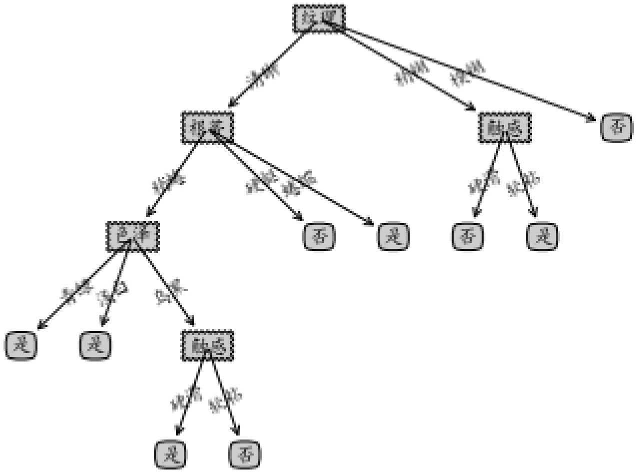
```
# 图形绘制
def createPlot(inTree):
    fig = plt.figure(1, facecolor='white')
    fig.clf()
    axprops = dict(xticks=[], yticks=[])
    createPlot.ax1 = plt.subplot(111, frameon=False, **axprops)    #no ticks
    #createPlot.ax1 = plt.subplot(111, frameon=False) #ticks for demo puropses
    plotTree.totalW = 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()
```

In [17]:

```
# 字体设置
matplotlib.rcParams['font.sans-serif']=['KaiTi']
matplotlib.rcParams['font.serif']=['KaiTi']
```

In [18]:

```
createPlot(node)
```



In [19]:

```
myTree={'no surfacing':{0:'no',1:{'flippers':{0:'no',1:'yes'}}}}
createPlot(myTree)
```