```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# @Time    : 2019/10/8 13:54
# @Author  : GXl
# @File    : 3.3.py
# @Software: win10 Tensorflow1.13.1 python3.5.6


from matplotlib.font_manager import FontProperties
import matplotlib.pyplot as plt
from math import log
import operator

"""
Parameters:
    dataSet - 数据集
Returns:
    shannonEnt - 经验熵(香农熵)
"""
# 函数说明:计算给定数据集的经验熵(香农熵)
def calcShannonEnt(dataSet):
    numEntires = len(dataSet)                        #返回数据集的行数
    labelCounts = {}                                 #保存每个标签(Label)
出现次数的字典
    for featVec in dataSet:                          #对每组特征向量进行
统计
        currentLabel = featVec[-1]                   #提取标签(Label)信
息
        if currentLabel not in labelCounts.keys(): #如果标签(Label)没有
放入统计次数的字典,添加进去
            labelCounts[currentLabel] = 0
        labelCounts[currentLabel] += 1               #Label 计数
    shannonEnt = 0.0                                 #经验熵(香农熵)
    for key in labelCounts:                          #计算香农熵
        prob = float(labelCounts[key]) / numEntires#选择该标签(Label)的
概率
        shannonEnt -= prob * log(prob, 2)            #利用公式计算
    return shannonEnt                                #返回经验熵(香农熵)

"""
Parameters:
    无
Returns:
    dataSet - 数据集
```

```
    labels - 特征标签
"""
# 函数说明:创建测试数据集
def createDataSet():
    file = open('lenses.data', 'r')
    files = file.readline()
    dataSet = []
    while files:
        line = list(map(str, files.split('  ')))
        list_ = []
        for i in range(1, 5):
            list_.append(int(line[i]))
        list_.append(str(line[5][0]))
        dataSet.append(list_)
        files = file.readline()
    #
    # dataSet = [[0, 0, 0, 0, 'no'],#数据集
    #         [0, 0, 0, 1, 'no'],
    #         [0, 1, 0, 1, 'yes'],
    #         [0, 1, 1, 0, 'yes'],
    #         [0, 0, 0, 0, 'no'],
    #         [1, 0, 0, 0, 'no'],
    #         [1, 0, 0, 1, 'no'],
    #         [1, 1, 1, 1, 'yes'],
    #         [1, 0, 1, 2, 'yes'],
    #         [1, 0, 1, 2, 'yes'],
    #         [2, 0, 1, 2, 'yes'],
    #         [2, 0, 1, 1, 'yes'],
    #         [2, 1, 0, 1, 'yes'],
    #         [2, 1, 0, 2, 'yes'],
    #         [2, 0, 0, 0, 'no']]
    labels = ['DD', 'A', 'E', 'R']#特征标签
    return dataSet, labels#返回数据集和分类属性

# def createDataSet():
#  # 数据集需要按照实验要求替换为指定
#     file = open('lenses.data', 'r')
#     a = file.read()
#     data = []
#     line = []
#     final = []
#     while a:
#         A, B, C = a.partition('\n')
#         data.append(A)
```

```
#         a = C
#     for each_line in data:
#         C = 1
#         while C:
#             A, B, C =


"""
Parameters:
    dataSet - 待划分的数据集
    axis - 划分数据集的特征
    value - 需要返回的特征的值
Returns:
    无
"""
# 函数说明:按照给定特征划分数据集
def splitDataSet(dataSet, axis, value):
    retDataSet = []                                     #创建返回的数据集列
表
    for featVec in dataSet:                             #遍历数据集
        if featVec[axis] == value:
            reducedFeatVec = featVec[:axis]             #去掉 axis 特征
            reducedFeatVec.extend(featVec[axis+1:])#将符合条件的添加到
返回的数据集
            retDataSet.append(reducedFeatVec)
    return retDataSet                                   #返回划分后的数据集

"""
Parameters:
    dataSet - 数据集
Returns:
    bestFeature - 信息增益最大的(最优)特征的索引值
"""
# 函数说明:选择最优特征
def chooseBestFeatureToSplit(dataSet):
    numFeatures = len(dataSet[0]) - 1                   #特征数量
    baseEntropy = calcShannonEnt(dataSet)                #计算数据集
的香农熵
    bestInfoGain = 0.0                                  #信息增益
    bestFeature = -1                                     #最优特征的
索引值
    for i in range(numFeatures):                         #遍历所有特
征
        #获取 dataSet 的第 i 个所有特征
```

```
        featList = [example[i] for example in dataSet]
        uniqueVals = set(featList)                                  #创建 set 集合{},元素不可重复
        newEntropy = 0.0                                            #经验条件熵
        for value in uniqueVals:                                    #计算信息增益
            subDataSet = splitDataSet(dataSet, i, value)   #subDataSet 划分后的子集
            prob = len(subDataSet) / float(len(dataSet))   #计算子集的概率
            newEntropy += prob * calcShannonEnt(subDataSet)#根据公式计算经验条件熵
        infoGain = baseEntropy - newEntropy                 #信息增益
        # print("第%d 个特征的增益为%.3f" % (i, infoGain))    #打印每个特征的信息增益
        if (infoGain > bestInfoGain):                       #计算信息增益
            bestInfoGain = infoGain                         #更新信息增益，找到最大的信息增益
            bestFeature = i                                 #记录信息增益最大的特征的索引值
    return bestFeature                                      #返回信息增益最大的特征的索引值

"""
Parameters:
    classList - 类标签列表
Returns:
    sortedClassCount[0][0] - 出现此处最多的元素(类标签)
"""
# 函数说明:统计 classList 中出现此处最多的元素(类标签)
def majorityCnt(classList):
    classCount = {}
    for vote in classList:#统计 classList 中每个元素出现的次数
        if vote not in classCount.keys():classCount[vote] = 0
        classCount[vote] += 1
    sortedClassCount     =     sorted(classCount.items(),     key     = operator.itemgetter(1), reverse = True)#根据字典的值降序排序
    return sortedClassCount[0][0]#返回 classList 中出现次数最多的元素

"""
Parameters:
    dataSet - 训练数据集
    labels - 分类属性标签
```

```
    featLabels - 存储选择的最优特征标签
Returns:
    myTree - 决策树
"""
# 函数说明:创建决策树
def createTree(dataSet, labels, featLabels):
    classList = [example[-1] for example in dataSet]        #取分类标签(是否放贷:yes or no)
    if classList.count(classList[0]) == len(classList):    #如果类别完全相同则停止继续划分
        return classList[0]
    if len(dataSet[0]) == 1:                                #遍历完所有特征时返回出现次数最多的类标签
        return majorityCnt(classList)
    bestFeat = chooseBestFeatureToSplit(dataSet)            #选择最优特征
    bestFeatLabel = labels[bestFeat]                        #最优特征的标签
    featLabels.append(bestFeatLabel)
    myTree = {bestFeatLabel:{}}                             #根据最优特征的标签生成树
    del(classList[bestFeat])                                    #删除已经使用特征标签
    featValues = [example[bestFeat] for example in dataSet]#得到训练集中所有最优特征的属性值
    uniqueVals = set(featValues)                            #去掉重复的属性值
    for value in uniqueVals:                                #遍历特征，创建决策树。
        myTree[bestFeatLabel][value] = createTree(splitDataSet(dataSet, bestFeat, value), labels, featLabels)
    return myTree


"""
Parameters:
    myTree - 决策树
Returns:
    numLeafs - 决策树的叶子结点的数目
"""
# 函数说明:获取决策树叶子结点的数目
def getNumLeafs(myTree):
    numLeafs = 0                                        #初始化叶子
    #python3中myTree.keys()返回的是dict_keys,不在是list,
    #所以不能使用 myTree.keys()[0]的方法获取结点属性，可以使用
```

```
list(myTree.keys())[0]
    firstStr = next(iter(myTree))
    secondDict = myTree[firstStr]                    #获取下一组字典
    for key in secondDict.keys():
        if  type(secondDict[key]).__name__=='dict':#测试该结点是否为字典，如果不是字典，代表此结点为叶子结点
            numLeafs += getNumLeafs(secondDict[key])
        else:   numLeafs +=1
    return numLeafs

"""
Parameters:
    myTree - 决策树
Returns:
    maxDepth - 决策树的层数
"""
# 函数说明:获取决策树的层数
def getTreeDepth(myTree):
    maxDepth = 0                                                #初始化决策树深度
    #python3中myTree.keys()返回的是dict_keys,不在是list,
    #所以不能使用 myTree.keys()[0]的方法获取结点属性，可以使用list(myTree.keys())[0]
    firstStr = next(iter(myTree))
    secondDict = myTree[firstStr]                               #获取下一个字典
    for key in secondDict.keys():
        if type(secondDict[key]).__name__=='dict':          #测试该结点是否为字典，如果不是字典，代表此结点为叶子结点
            thisDepth = 1 + getTreeDepth(secondDict[key])
        else:   thisDepth = 1
        if thisDepth > maxDepth: maxDepth = thisDepth     #更新层数
    return maxDepth

"""
Parameters:
    nodeTxt - 结点名
    centerPt - 文本位置
    parentPt - 标注的箭头位置
    nodeType - 结点格式
Returns:
    无
"""
# 函数说明:绘制结点
```

```
def plotNode(nodeTxt, centerPt, parentPt, nodeType):
    arrow_args                  =                  dict(arrowstyle="<-")
#定义箭头格式
    #   font   =   FontProperties(fname=r"c:\windows\fonts\simsun.ttc",
size=14)    #设置中文字体
    createPlot.ax1.annotate(nodeTxt,   xy=parentPt,      xycoords='axes
fraction',#绘制结点
        xytext=centerPt, textcoords='axes fraction',
        va="center", ha="center", bbox=nodeType, arrowprops=arrow_args)

"""
Parameters:
    cntrPt、parentPt - 用于计算标注位置
    txtString - 标注的内容
Returns:
    无
"""
# 函数说明:标注有向边属性值
def plotMidText(cntrPt, parentPt, txtString):
    xMid = (parentPt[0]-cntrPt[0])/2.0 + cntrPt[0]#计算标注位置
    yMid = (parentPt[1]-cntrPt[1])/2.0 + cntrPt[1]
    createPlot.ax1.text(xMid, yMid, txtString, va="center", ha="center",
rotation=30)

"""
Parameters:
    myTree - 决策树(字典)
    parentPt - 标注的内容
    nodeTxt - 结点名
Returns:
    无
"""
# 函数说明:绘制决策树
def plotTree(myTree, parentPt, nodeTxt):
    decisionNode      =      dict(boxstyle="sawtooth",      fc="0.8")
#设置结点格式
    leafNode        =        dict(boxstyle="round4",        fc="0.8")
#设置叶结点格式
    numLeafs                    =                    getNumLeafs(myTree)
#获取决策树叶结点数目，

                    #决定了树的宽度
    depth                    =                    getTreeDepth(myTree)
#获取决策树层数
```

```
    firstStr = next(iter(myTree))
#下个字典
    cntrPt = (plotTree.xOff + (1.0 + float(numLeafs))/2.0/plotTree.totalW, plotTree.yOff)#中心位置
    plotMidText(cntrPt, parentPt, nodeTxt)
#标注有向边属性值
    plotNode(firstStr, cntrPt, parentPt, decisionNode)
#绘制结点
    secondDict = myTree[firstStr]
#下一个字典，

                    #也就是继续绘制子结点
    plotTree.yOff = plotTree.yOff - 1.0/plotTree.totalD
#y 偏移
    for key in secondDict.keys():
        if type(secondDict[key]).__name__=='dict':
#测试该结点是否为字典，

                    #如果不是字典，

                    #代表此结点为叶子结点
            plotTree(secondDict[key],cntrPt,str(key))
#不是叶结点，

                    #递归调用继续绘制
        else:
#如果是叶结点，绘制叶结点，

#并标注有向边属性值
            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

"""
Parameters:
    inTree - 决策树(字典)
Returns:
    无
"""
# 函数说明:创建绘制面板
def createPlot(inTree):
```

```
    fig = plt.figure(1, facecolor='white')                                  #创建
fig
    fig.clf()                                                               #清空
fig
    axprops = dict(xticks=[], yticks=[])
    createPlot.ax1 = plt.subplot(111, frameon=False, **axprops)#去掉 x、
y 轴
    plotTree.totalW = float(getNumLeafs(inTree))                            #获取决
策树叶结点数目
    plotTree.totalD = float(getTreeDepth(inTree))                           #获取决
策树层数
    plotTree.xOff = -0.5/plotTree.totalW; plotTree.yOff = 1.0; #x 偏移
    plotTree(inTree, (0.5,1.0), '')                                         #绘制决
策树
    plt.show()                                                              #显示绘
制结果


if __name__ == '__main__':
    dataSet, labels = createDataSet()
    featLabels = []
    myTree = createTree(dataSet, labels, featLabels)
    print(myTree)
    createPlot(myTree)
```