实验二决策树分类器

一、实验目的及要求   
 1、掌握 Python 安装、编程环境搭建的法。

2、熟练掌握 matplotlib 中注释函数的用法。

3、掌握决策树分类器的基本工作原理。

4、掌握利用 matplotlib 中注释函数进行决策树绘制的方法

二、预习要求   
 阅读本实验例程部分，实现基本的决策树分类算法以及决策树绘制方法，以便能够

充分利用实验时间编程调试。

三、实验设备   
 硬件：PC机。

软件：Python 及相关集成开发环境。

四、实验内容   
利用利用 Python 编程实现基于决策树分类算法，并将训练得到决策树绘制来。

具体要求如下：

1. 利用给定的贷款信息数据构造贷款决策树，并进行简单的测试
2. 将构建的贷款决策树绘制出来

（3）利用给定的隐形眼镜数据，构建是镜片软硬以及是否戴镜的决策树，并进行简单

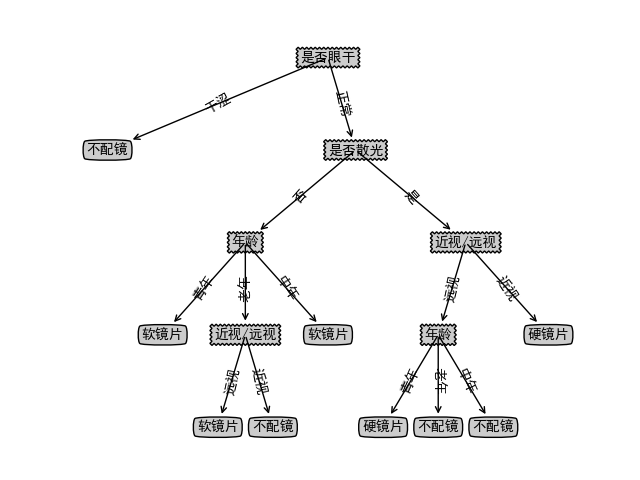
测试

（4）将隐形眼镜决策树绘制出来，并对测试结果进行简单分析。

决策树绘制部分参考视频：https://www.bilibili.com/video/BV1gc411j7oX/

1. 实验报告内容要求   
   1. 列出编写的 python 代码。对主要的语句进行注释

贷款决策树：

from matplotlib import pyplot as plt  
import numpy as np  
*决策节点*decisionNode = dict(boxstyle=**"sawtooth"**, fc=**"0.8"**)  
*叶节点*leafNode = dict(boxstyle=**"round4"**, fc=**"0.8"**)  
*箭头、分支*arrow\_args = dict(arrowstyle=**"<-"**)  
  
class PlotTree:  
 def \_\_init\_\_(self,inTree,ax):  
  
 *获取树的宽度和高度* self.inTree = inTree  
 self.totalW = float(self.\_getNumLeafs(inTree))  
 self.totalD = float(self.\_getTreeDepth(inTree))  
  
 *设置初始的偏移量* self.xOff = -0.5/self.totalW   
 self.yOff = 1.0  
 self.ax = ax   
  
  
 def \_getNumLeafs(self,myTree):  
 numLeafs = 0  
 keys = myTree.keys()  
 firstStr = list(keys)[0]  
 secondDict = myTree[firstStr]  
 for key in secondDict.keys():  
 if type(secondDict[key]).\_\_name\_\_==**'dict'**:  
 numLeafs += self.\_getNumLeafs(secondDict[key])  
 else: numLeafs +=1  
 return numLeafs  
   
 def \_getTreeDepth(self,myTree):  
 maxDepth = 0  
 keys = list(myTree.keys())  
 firstStr = keys[0]  
 secondDict = myTree[firstStr]  
 for key in secondDict.keys():  
 if type(secondDict[key]).\_\_name\_\_==**'dict'**:  
 thisDepth = 1 + self.\_getTreeDepth(secondDict[key])  
 else: thisDepth = 1  
 if thisDepth > maxDepth:   
 maxDepth = thisDepth  
 return maxDepth  
   
 def \_plotNode(self,nodeTxt, centerPt, parentPt, nodeType):  
 self.ax.annotate(nodeTxt, xy=parentPt, xycoords=**'axes fraction'**,  
 xytext=centerPt, textcoords=**'axes fraction'**,  
 va=**"center"**, ha=**"center"**, bbox=nodeType, arrowprops=arrow\_args )  
   
  
 def \_plotMidText(self,cntrPt, parentPt, txtString):  
 *获取的中点* xMid = (parentPt[0]+cntrPt[0])/2.0  
 yMid = (parentPt[1]+cntrPt[1])/2.0  
   
 *计算、连线与水平方向的夹角* if parentPt[0]-cntrPt[0] ==0:  
 theta =90   
 else:  
 theta = np.arctan((parentPt[1]-cntrPt[1])/(parentPt[0]-cntrPt[0]))\*180/np.pi  
   
 self.ax.text(xMid, yMid, txtString, va=**"center"**, ha=**"center"**, rotation=theta)  
  
 *当前树  
 父节点的位置  
 指向当前树的文字* def \_plotTree(self,myTree, parentPt, nodeTxt):  
 *获取当前树的所有叶子节点的数目，即当前树的宽度* numLeafs = self.\_getNumLeafs(myTree)   
   
 keys = list(myTree.keys())  
 firstStr = keys[0]   
  
 *当前节点的位置应该在所有当前树的中间* cntrPt = (self.xOff + (1.0 + float(numLeafs))/2.0/self.totalW, self.yOff)  
 self.\_plotMidText(cntrPt, parentPt, nodeTxt)  
 plt.pause(1)  
 self.\_plotNode(firstStr, cntrPt, parentPt, decisionNode)  
 plt.pause(1)  
 secondDict = myTree[firstStr]  
 *每画深一层减少* self.yOff = self.yOff - 1.0/self.totalD  
 for key in secondDict.keys():  
 *下一层是字典 画树* if type(secondDict[key]).\_\_name\_\_==**'dict'**:self.\_plotTree(secondDict[key],cntrPt,str(key)) *下一层是叶子* else:  
 *每画一个叶子增加* self.xOff = self.xOff + 1.0/self.totalW  
 self.\_plotNode(secondDict[key], (self.xOff, self.yOff), cntrPt, leafNode)  
 plt.pause(1)  
 self.\_plotMidText((self.xOff, self.yOff), cntrPt, str(key))  
 plt.pause(1)  
 *返回一层增加* self.yOff = self.yOff + 1.0/self.totalD  
  
 def draw(self):  
 self.\_plotTree(self.inTree, (0.5,1.0), **''**)  
 plt.show()  
  
  
if \_\_name\_\_ ==**"\_\_main\_\_"**:  
  
  
 in\_tree = {**'no surfacing'**: {0: **'no'**, 1: {**'flippers'**: {0: **'no'**, 1: **'yes'**}}}}  
 cn\_tree= {**'是否眼干'**: {**'干涩'**: **'不配镜'**, **'正常'**: {**'是否散光'**: {**'否'**: {**'年龄'**: {**'青年'**: **'软镜片'**, **'老年'**: {**'近视/远视'**: {**'远视'**: **'软镜片'**, **'近视'**: **'不配镜'**}}, **'中年'**: **'软镜片'**}}, **'是'**: {**'近视/远视'**: {**'远视'**: {**'年龄'**: {**'青年'**: **'硬镜片'**, **'老年'**: **'不配镜'**, **'中年'**: **'不配镜'**}}, **'近视'**: **'硬镜片'**}}}}}}  
 *画布布局* plt.rcParams[**'font.sans-serif'**]=[**'SimHei'**]  
 plt.rcParams[**'axes.unicode\_minus'**]=False  
 fig = plt.figure(1, facecolor=**'white'**)  
 fig.clf()  
 axprops = dict(xticks=[], yticks=[])  
 ax = plt.subplot(111, frameon=False, \*\*axprops)  
 m\_plotTree = PlotTree(cn\_tree,ax=ax)  
 m\_plotTree.draw()  
实验结果：  
  


from math import log  
import operator  
import pickle  
from new\_drawTree import PlotTree  
from matplotlib import pyplot as plt  
from new\_drawTree import PlotTree  
  
*创建数据集*def createDataSet():  
 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 = [**'年龄'**, **'有工作'**, **'有自己的房子'**, **'信贷情况'**] *特征标签* return dataSet, labels *返回数据集和分类属性  
  
  
计算经验熵香农熵*def calcShannonEnt(dataSet):  
 *返回数据集的行数* numEntires = len(dataSet)  
  
 *收集所有目标标签 （最后一个维度）* labels = [featVec[-1] for featVec in dataSet]  
  
 *去重、获取标签种类* keys = set(labels)  
  
 shannonEnt = 0.0  
 for key in keys:  
 *计算每种标签出现的次数* prob = float(labels.count(key)) / numEntires  
 shannonEnt -= prob \* log(prob, 2)  
 return shannonEnt  
  
  
*数据集分割  
将第维 等于的数据集提取出来*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*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  
 获取的第个所有特征* featList = [example[i] for example in dataSet]  
 uniqueVals = set(featList) *创建集合元素不可重复* newEntropy = 0.0 *经验条件熵* for value in uniqueVals: *计算信息增益* subDataSet = splitDataSet(dataSet, i, value) *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  
 第个特征的增益为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  
  
  
返回中出现次数最多的元素*def majorityCnt(classList):  
 classCount = {}  
 keys = set(classList)  
 for key in keys:  
 classCount[key] = classList.count(key)  
  
 *根据字典的值降序排序* sortedClassCount = sorted(classCount.items(),  
 key=operator.itemgetter(1),  
 reverse=True)  
 return sortedClassCount[0][0]  
  
  
*创建决策树*def createTree(dataSet, labels, lab\_sel):  
 *取分类标签是否放贷*classList = [example[-1] for example in dataSet]  
  
 *如果类别完全相同则停止继续划分* if classList.count(classList[0]) == len(classList):  
 return classList[0]  
  
 *遍历完所有特征时返回出现次数最多的类标签* if len(dataSet[0]) == 1 or len(labels) == 0:  
 return majorityCnt(classList)  
  
 *获取最优特征的维度* bestFeat = chooseBestFeatureToSplit(dataSet)  
  
 *得到最优特征的标签* bestFeatLabel = labels[bestFeat]  
 lab\_sel.append(labels[bestFeat])  
  
 *根据最优特征的标签生成树* myTree = {bestFeatLabel: {}}  
  
 *删除已经使用特征标签* del (labels[bestFeat])  
  
 *得到训练集中所有最优特征维度的所有属性值* featValues = [example[bestFeat] for example in dataSet]  
 uniqueVals = set(featValues) *去掉重复的属性值* for value in uniqueVals: *遍历特征，创建决策树。* subLabels = labels[:]  
 myTree[bestFeatLabel][value] = createTree(splitDataSet(dataSet, bestFeat, value), subLabels, lab\_sel)  
  
 return myTree  
  
  
*进行分类*def classify(inputTree, featLabels, testVec):  
 firstStr = next(iter(inputTree)) *获取决策树结点* secondDict = inputTree[firstStr] *下一个字典* featIndex = featLabels.index(firstStr)  
 for key in secondDict.keys():  
 if testVec[featIndex] == key:  
 if type(secondDict[key]).\_\_name\_\_ == **'dict'**:  
 classLabel = classify(secondDict[key], featLabels, testVec)  
 else:  
 classLabel = secondDict[key]  
 return classLabel  
  
  
if \_\_name\_\_ == **'\_\_main\_\_'**:  
 *获取数据集* dataSet, labels = createDataSet()  
 lab\_copy = labels[:]  
 lab\_sel = []  
 myTree = createTree(dataSet, labels,lab\_sel)  
 print(myTree)  
 print(lab\_sel)  
 *测试* testVec = [0,1,1,2]  
 result = classify(myTree,lab\_copy,testVec)  
 print(result)  
  
 *画布布局* plt.rcParams[**'font.sans-serif'**] = [**'SimHei'**]  
 plt.rcParams[**'axes.unicode\_minus'**] = False  
 fig = plt.figure(1, facecolor=**'white'**)  
 fig.clf()  
 axprops = dict(xticks=[], yticks=[])  
 ax = plt.subplot(111, frameon=False, \*\*axprops)  
 m\_plotTree = PlotTree(myTree, ax=ax)  
 m\_plotTree.draw()  
  
 *年龄近视远视是否散光是否眼干测试  
 输入特征：预测结果医生推荐*

