

**计算机与信息 学院实验报告**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 实验课程： | 人工智能实验 | | | | |
| 实验编号： | 实验十 | | | | |
| 实验名称： | 机器学习 | | | | |
| 实验人员： | 学号 | 18111207248 | | | |
| 姓名 | 吴钰 | | | |
| 班级 | 18级计算机科学与技术（创新班） | | | |
| 实验日期： | 2020.12.10 | | | | |
| 实验室： | 2060402 | | | | |
|  |  | | | | |
| 实验评价： |  | | | | |
| 实验成绩： | |  | 评价日期： |  |
|  | 指导教师： | |  | | |

# 实验目的

## 编程实现书本第10章P289页编程题1或编程题4（两题选做一题），完成并提交实验报告。

# 二、实验要求

## 完成实验后请填写实验报告并上交。要求在实验报告中必须记录实验中遇到的问题及其问题解决方案。

# 三、实验内容

选择编程题1

源代码：

# -\*- coding: utf-8 -\*-

"""

Created on Fri Dec 18 15:57:23 2020

@author: wuyu

"""

# -\*- coding: utf-8 -\*-

"""

Created on Fri Dec 18 08:43:19 2020

@author: TBABMB

"""

from math import log

import operator

import matplotlib.pyplot as plt

from matplotlib.font\_manager import FontProperties

from matplotlib.font\_manager import FontProperties

import matplotlib.pyplot as plt

def createDataSet():#创建样本数据

"""

酱汁颜色：0代表红色 1代表粉色 2代表黄色

含肉： 1代表是 0代表否

含海鲜： 1代表真 0代表假

喜欢： Y表示是 N表示否

"""

dataSet=[[0,1,0,'Y'],#数据集

[0,0,0,'N'],

[0,0,1,'N'],

[2,0,1,'N'],

[1,0,0,'Y'],

[0,1,0,'Y'],

[2,0,0,'Y'],

[2,0,0,'Y'],

[0,0,0,'N'],

[1,0,0,'Y']

]

labels=['颜色','含肉','含海鲜']#分类属性

return dataSet,labels #返回数据集和分类属性

def calcShannonEnt(dateSet): # 计算经验熵

numEntires = len(dateSet) #数据集行数

labelCounts={}

for featVec in dateSet:

currentLabel = featVec[-1]

if currentLabel not in labelCounts.keys():

labelCounts[currentLabel] = 0

labelCounts[currentLabel] += 1

shannonEnt = 0.0

for key in labelCounts:

prob = float(labelCounts[key]) / numEntires #选择该标签(Label)的概率

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

return shannonEnt #返回经验熵(香农熵)

"""

按照给定特征划分数据集

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

"""

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

"""

选择最优特征

Parameters:

dataSet - 数据集

Returns:

bestFeature - 信息增益最大的(最优)特征的索引值

"""

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

"""

统计classList中出现此处最多的元素(类标签)

Parameters:

classList - 类标签列表

Returns:

sortedClassCount[0][0] - 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

"""

递归构建决策树

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

"""

def createTree(dataSet, labels, featLabels):

classList = [example[-1] for example in dataSet] #取分类标签(是否喜欢: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(labels[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:

subLabels=labels[:]

#递归调用函数createTree(),遍历特征，创建决策树。

myTree[bestFeatLabel][value] = createTree(splitDataSet(dataSet, bestFeat, value), subLabels, featLabels)

return myTree

"""

使用决策树执行分类

Parameters:

inputTree - 已经生成的决策树

featLabels - 存储选择的最优特征标签

testVec - 测试数据列表，顺序对应最优特征标签

Returns:

classLabel - 分类结果

"""

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

#定义文本框和箭头格式

decisionNode=dict(boxstyle='sawtooth',fc='0.8')

leafNode=dict(boxstyle='round4',fc='0.8')

arrow\_args=dict(arrowstyle='<-')

#设置中文字体

font = FontProperties(fname=r"c:\windows\fonts\simsun.ttc", size=14)

"""

获取决策树叶子结点的数目

Parameters:

myTree - 决策树

Returns:

numLeafs - 决策树的叶子结点的数目

"""

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

"""

获取决策树的层数

Parameters:

myTree - 决策树

Returns:

maxDepth - 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

"""

绘制结点

Parameters:

nodeTxt - 结点名

centerPt - 文本位置

parentPt - 标注的箭头位置

nodeType - 结点格式

"""

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,fontproperties=font)

"""

标注有向边属性值

Parameters:

cntrPt、parentPt - 用于计算标注位置

txtString - 标注的内容

"""

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)

"""

绘制决策树

Parameters:

myTree - 决策树(字典)

parentPt - 标注的内容

nodeTxt - 结点名

"""

def plotTree(myTree, parentPt, nodeTxt):

decisionNode = dict(boxstyle="round4", fc="1") #设置结点格式

leafNode = dict(boxstyle="sawtooth", 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

"""

创建绘制面板

Parameters:

inTree - 决策树(字典)

"""

def createPlot(inTree):

fig = plt.figure(1, facecolor='pink') #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()

dataSet, labels = createDataSet()

featLabels = []

myTree = createTree(dataSet, labels, featLabels)

print(myTree)

testVec = [0, 1] # 测试数据

result = classify(myTree, featLabels, testVec)

if result == 'yes':

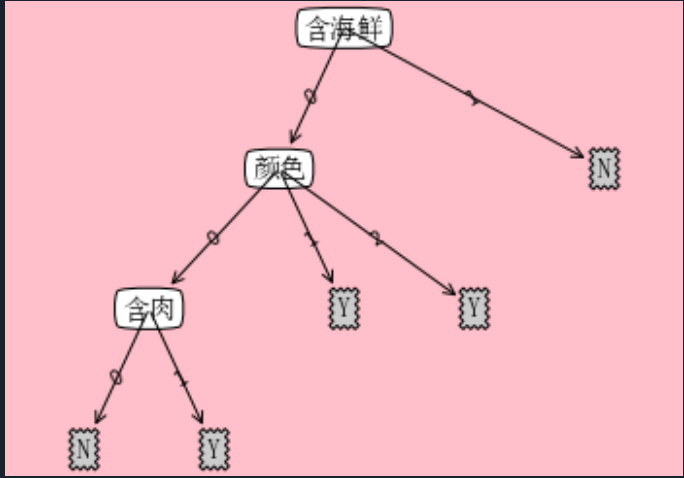
print('喜欢')

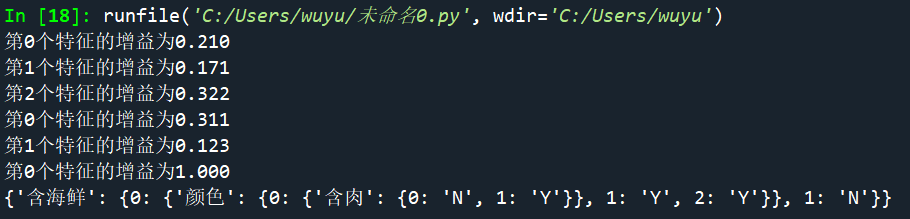
elif result == 'no':

print('不喜欢')

createPlot(myTree)

运行截图：





# 四、实验思考

1、决策树模型是运用于分类以及回归的一种树结构。决策树由节点和有向边组成，一般一棵决策树包含一个根节点、若干内部节点和若干叶节点。决策树的决策过程需要从决策树的根节点开始，待测数据与决策树中的特征节点进行比较，并按照比较结果选择选择下一比较分支，直到叶子节点作为最终的决策结果。

* 内部节点：对应于一个属性测试
* 叶节点：对应于决策结果
* 根节点包含样本全集；
* 每个节点包括的样本集合根据属性测试的结果被划分到子节点中；
* 根节点到每个叶节点的路径对应对应了一个判定测试路径；

2、特征选择也即选择最优划分属性，从当前数据的特征中选择一个特征作为当前节点的划分标准。我们希望在不断划分的过程中，决策树的分支节点所包含的样本尽可能属于同一类，即节点的“纯度”越来越高。而选择最优划分特征的标准不同，也导致了决策树算法的不同。

为了找到最优的划分特征，我们需要先了解一些信息论的知识：

* 信息熵(*information entropy*)
* 信息增益(*information gain*)
* 信息增益率(*information gain ratio*)
* 基尼指数(*Gini index*)