电子科技大学

计算机科学与工程学院

实 验 指 导 书

（实验）课程名称 人工智能综合实验 I

电子科技大学教务处制表

实验六　决策树（Decision Tree）实现目标分类

一、 实验目的和任务

本实验目的通过代码实现决策树学习算法，编程实践中在 Python 中使用 Matplotlib 库注解绘制树形图，并能够将算法用于目标分类。通过本实验，希望能够掌握理决策树模型与学习基本概念，并能够将算法用于执行目标分类问题。

二、 实验原理

决策树是机器学习中经常使用的分类方法，也是一种经常使用的监督学习方法，首先对数据进行处理，利用归纳算法生成可读的规则和决策树，然后使用决策对新数据进行分析。本质上决策树是通过一系列规则对数据进行分类的过程。决策树技术发现数据模式和规则的核心是归纳算法。

决策树的基本组成部分：决策结点、分支和叶子。决策树中最上面的结点称为根结点。是整个决策树的开始。每个分支是一个新的决策结点，或者是树的叶子。每个决策结点代表一个问题或者决策。通常对应待分类对象的属性。每个叶结点代表一种可能的分类结果。决策树学习的目的就是为了构造一棵泛化能力强，即处理待测样本能力强的决策树，基本算法遵循自顶向下、分而治之的策略。

1. 决策树 CLS 算法步骤：

（1）生成一颗空决策树和一张训练样本属性集;

（2）若训练样本集 T 中所有的样本都属于同一类, 则生成叶结

点 T，并终止学习算法; 否则

(3) 根据某种策略从训练样本属性表中选择属性 A 作为测试属性, 生成测试结点 A;

(4) 若 A 的取值为 v1, v2,…, vm, 则根据 A 的取值的不同,将 T 划分成 m 个子集 T1, T2,…, Tm;

(5) 从训练样本属性表中删除属性 A;

转步骤（2），对每个子集递归调用 CLS。

2. 最佳划分的度量问题

随着长树过程的不断进行，我们希望决策树的分支结点所包含的样本越来越归属于同一类别，即结点的“不纯度” (impurity) 越来越低。因此，为了确定按某个属性划分的效果，我们需要比较划分前（父亲结点）和划分后（所有子结点）不纯度的降低程度，降低越多，划分的效果就越好。

在实验中，可选用以下 3 种最佳划分的度量标准：

（1）熵减最大：$\Delta_{Entropy\,\mathrm{Re}\,duction} = Entropy(parent) - \sum_{j=1}^{k} \frac{N(j)}{N} Entropy(j)$

（2）基尼指数减最大：$\Delta_{Gini\,\mathrm{Re}\,duction} = Gini(parent) - \sum_{j=1}^{k} \frac{N(j)}{N} Gini(j)$

（3）误分类率减最大：$\Delta_{Error\,\mathrm{Re}\,duction} = Error(parent) - \sum_{j=1}^{k} \frac{N(j)}{N} Error(j)$

3. 叶子结点的判定

如果我们暂且不考虑树的规模过大而导致的过拟合问题，在决策树学习基本算法中，有三种情形会判定为叶子结点:

(1) 当前结点中的样本集合为空，即空叶子；

(2) 前结点中的所有样本全部归属于同一类别，即纯叶子；

(3) 当前结点中的所有样本在所有属性上取值相同，即属性被测试完的叶子。

4. 剪枝问题

剪枝 (pruning)是决策树学习算法对付“过拟合”的主要手段。是否剪枝，取决于剪枝能否带来决策树泛化性能提升。剪枝又分为预剪枝和后剪枝两种。是否剪枝，取决于剪枝前后度量结果。

5. 树结构可视化

决策树的结构是树形结构，目前得到的树的表示比较抽象，不直观。这里我们将使用 matplotlib 数据包进行树的可视化。可视化这样一颗决策树据需设计树的分支节点和叶子结点的分布，这将直接决定可视化树的长宽。

6. Python 函数准备

（1）由给定的数据集文件，进行数据和标签的读取，并转化为数组形式输出

read_csv( )或 f=open( ) + f.readlines( ).

（2）其他函数

map(func,*iterable) #根据提供的函数对指定的序列做映射。通俗地讲就是以参数序列中的每个元素分别调用参数中的函数（func（）），把每次调用后返回的结果保存到返回值中

Sorted (iterable, key=None, reverse=False) #其中，iterable 表示指定的序列，key 参数可以自定义排序规则；reverse 参数指定以升序（False，默认）还是降序（True）进行排序。sorted() 函数会返回一个排好序的列表。

operator.itemgetter(*items) #返回一个可调用对象，用于从运算对象中获取元素，例如

a = operator.itemgetter(1), a('jinagioa')>> 'i'

（3）画图函数 import matplotlib.pyplot as plt

import operator

plt.savefig(‘test’, dpi = 600)　#将绘制的图画保存成 png 格式，命名为 test

plt.ylabel(‘Grade’)　#y 轴的名称

plt.axis([-1, 10, 0, 6])　#x 轴起始于-1，终止于 10 ，y 轴起始于 0，终止于 6

plt.subplot(3,2,4)　#分成 3 行 2 列，共 6 个绘图区域，在第 4 个区域绘图。排序为行优先。也可 plt.subplot(324)，将逗号省略。

plt.plot(x, y, format_string, **kwargs)　#x 为 x 轴数据，可为列表或数组；y 同理；format_string 为控制曲线的格式字符串， **kwargs 第二组或更多的（x, y, format_string）

plt.xlabel(‘横轴：时间’, fontproperties = ‘simHei’, fontsize = 20)　#对 x 轴增加文本标签

plt.ylabel( ) #同理

7. 实验 demo 参考

下面我们会给出实验的总体 demo，部分算法需要同学们自己补充。

首先进行数据集的创建，由给定的数据集文件，进行数据和标签的读取，并转化为数组形式输出

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

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()

    labels = ['DD', 'A', 'E', 'R']          #特征标签
    return dataSet, labels              #返回数据集和分类属性
```

创建决策树：

```
"""
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
"""
# 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

"""
```

训练过程中我们每次要根据信息增益从剩余的特征属性中选出最好的哪个属性，最好的标准由我们所选择的评价指标来定，选择最优特征属性的代码如下

```
"""
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
```

在属性选择创建新节点和分枝的过程中，需要依据信息增益判断是否新增节点和分枝，以及预剪枝和后剪枝等操作。

下面函数用来计算信息增益，这里给出 Gini 指标的计算函数，自己书写信息熵和分类率的计算函数。

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


  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

def calcGini(dataSet):
        labels_count={}
        number=len(dataSet)
        for i,data in enumerate(dataSet):
            label=dataSet[i][-1]
            if label in labels_count.keys():
                labels_count[label]+=1;
            else:
                labels_count[label]=1;
        Gini=0.0;
        for label,value in labels_count.items():
            pr=1.0 * value / number * value / number
            Gini += 1 - pr
        return Gini
```

构建决策树时，先根据当前的所有可用属性选出最优属性，然后按照该属性划分子数据集和分枝，之后在新的属性节点递归该过程，直到子集样本的类别相同或者子集样本的取值一样时停止，节点标签设为样本标签最多的那个。

划分子集的代码如下

```
    """
    函数说明：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：需要返回的特征值
    Returns：
        返回划分后的数据集
    """
    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
```

可视化决策树的整体代码与相关函数

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



"""
函数说明：获取决策树叶子节点的数目
Parameters：
    myTree：决策树
Returns：
    numLeafs：决策树的叶子节点的数目
"""
def getNumLeafs(myTree):
    numLeafs=0
    firstStr=next(iter(myTree))
    secondDict=myTree[firstStr]
    for key in secondDict.keys():
        if type(secondDict[key]).__name__=='dict':
            numLeafs+=getNumLeafs(secondDict[key])
        else: numLeafs+=1
    return numLeafs
```

```
"""
函数说明: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
    firstStr = next(iter(myTree))                                   #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]
    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



"""
函数说明: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
Returns:
    无
"""
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)



"""
函数说明: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
    Returns:
        无
    """
    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)



    """
    函数说明:绘制决策树

    Parameters:
        myTree - 决策树(字典)
        parentPt - 标注的内容
        nodeTxt - 结点名
    Returns:
        无
    """
    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
```

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

三、 实验内容

1. 用 python 语言依据给出的 demo 代码完成决策树算法的实现，并对下面 2 个数据集进行分类：

1） LENSES 隐形眼镜数据集；

数据集一共 24 个样本，3 个类别，每一个样本有 4 个特征：age、spectacle prescription、astigmatic、tear production rate。数据文件存储规则为 24×6，第一列为样本序号，第 2~5 列为 4 个特征值，最后一列为样本所属类别。具体特征对应的取值以及含义请查看附件

lenses.names 中的详细介绍，数据文件为 lenses.data。

2. 对上述数据集构造的决策树进行可视化，分别给出可视化结果。

四、 实验报告要求

根据实验内容按步骤完成实验，并给出数据集的分类准确率（包括训练集和测试集，训练集和测试集可以自行划定）。给出数据集构造的决策树的可视化结果图。分析实验结果，总结实验收获和经验。

五、 实验仪器设备

机房电脑一台，编程平台为 Anaconda 下 Spyder 编辑器。