1. **自己提出的问题的理解：**

**问题1：**为什么说熵只和分布有关，而和值无关。

讨论后的理解：根据熵的公式可以知道，熵的值由随机变量中各个属性的值的概率所决定，而属性的概率值由随机变量所属的分布所决定，而随机变量的值和属性的概率值无关，所以熵只和分布有关，而和随机变量的值无关。

根据熵的意义可以知道它是指事物的无序度，而一个随机变量的无序的程度只和它的分布有关，和其具体的值并不相关，所以从这个方面也可以得出熵只和分布有关而和值无关。

**问题2：**P79页为什么C（T）可以表示模型对训练数据的预测误差，它的数学意义是什么？

讨论后的理解：首先经验熵可以描述一个随机变量某个特征的无序度，C（T）的公式求当前类别的样本数乘上它的经验熵，而样本数则表示有多少个样本具有当前的经验熵，而经验熵的无序度能够表现出当前决策树在该类别上的预测误差，样本数的累积则是决策树在该类别上的累计预测误差。

1. **别人提出的问题的理解：**

**问题1：**以信息增益作为划分训练数据集的特征，为什么会存在偏向于选取值较多的特征？

讨论后的理解：根据信息增益的公式可以看出来，在求信息增益的时候它的值和当前特征的属性取值个数相关，而取值个数越多的话，因为最后信息增益的结果是求相加之后的值，所以会随着取值个数的增多而增加，所以会偏向于选择取值比较多的特征。

**问题2：**cart算法对比C4.5算法有什么优势，为什么要选择cart算法？

讨论后的理解：

* cart算法既可以用来进行分类，又可以用来进行回归，而C4.5则只能用来分类。
* C4.5算法在过程中需要对数据进行多此排序，所以不适合数据量比较大的情况，而CART则一般适用于大数据的情况，而CART对小样本则泛化能力比较差。
* C4.5是通过剪枝来提高决策树的准确性，防止过拟合，而CART则是通过全部数据来发现所有树的结构进行对比，
* C4.5只能单次使用特征变量，而CART则可以重复使用。

**问题3：**P81为什么说Rm上的最优值是Rm上所有输入实例x对应的y的均值？

讨论后的理解：根据平方误差公式可以看出，为了使得最后的预测误差能够尽可能的小，需要平方误差尽可能小，即Rm上的输出值cm的值与样本的真实值能够尽可能相近。为了达到这个目标，则需要该单元上的所有样本点到达cm的距离和是最小的，而只有当cm是该单元上所有样本点的输出的均值的时候，此时平方误差最小，所以Rm上的最优值应该是Rm上所有输入实例x对应的y的均值。

1. **读书计划**

1、本周完成的内容章节：《统计学习方法》第5章

2、下周计划：《统计学习方法》第6章

1. **读书摘要及理解或伪代码的具体实现**
2. **读书摘要：**
   1. 决策树的内部节点表示规则，叶结点表示分的类。
   2. 一个实例只有一条路径。
   3. 决策树的生成只考虑局部最优，相对地，决策树的剪枝则考虑全局最优。
   4. 熵只与分布有关，而和值无关，因为熵的取值依赖于概率值，而概率是和随机变量的分布相关，而和其具体的值无关。
   5. 当熵和条件熵中的概率由数据估计(特别是极大似然估计)得到时，所对应的熵与条件熵分别称为经验熵和经验条件熵。
   6. 决策树学习中的信息增益等价于训练数据集中类与特征的互信息（熵 H(Y) 与条件熵 H(YIX) 之差称为互信息）。
   7. 有三种情况下会停止递归变成叶结点：1）D中所有实例属于同一类 2）没有特征了 3）信息增益都小于阈值
   8. ID3和C4.5的区别为计算信息增益的时候的方法，一个是计算信息增益的值，一个是信息增益比。
   9. 决策树剪枝可以通过减小模型复杂度来减小过拟合程度
3. **代码实现**

#决策树

from math import log

import operator

def CalEnt(dataset):

    numEntries=len(dataset)

    labelCounts={}

    for featVec in dataset:

        currentlabel=featVec[-1]

        if currentlabel not in labelCounts.keys():

            labelCounts[currentlabel]=0

        labelCounts[currentlabel]+=1

    Ent=0.0

    for key in labelCounts:

        p=float(labelCounts[key])/numEntries

        Ent=Ent-p\*log(p,2)

    return Ent

def splitdataset(dataset,axis,value):

    retdataset=[]

    for featVec in dataset:

        if featVec[axis]==value:

            reducedfeatVec=featVec[:axis]

            reducedfeatVec.extend(featVec[axis+1:])

            retdataset.append(reducedfeatVec)

    return retdataset

def ChooseFeatures(dataset):

    numFeatures = len(dataset[0]) - 1

    bestGini = 100000.0

    bestFeature = -1

    for i in range(numFeatures):

        featList = [example[i] for example in dataset]

        uniqueVals = set(featList)

        gini = 0.0

        for value in uniqueVals:

            subdataset=splitdataset(dataset,i,value)

            p=len(subdataset)/float(len(dataset))

            subp = len(splitdataset(subdataset, -1, '0')) / float(len(subdataset))

        gini += p \* (1.0 - pow(subp, 2) - pow(1 - subp, 2))

        if (gini < bestGini):

            bestGini = gini

            bestFeature = i

    return bestFeature

def majorityCnt(classList):

    classCont={}

    for vote in classList:

        if vote not in classCont.keys():

            classCont[vote]=0

        classCont[vote]+=1

    sortedClassCont=sorted(classCont.items(),key=operator.itemgetter(1),reverse=True)

    return sortedClassCont[0][0]

def CART\_createTree(dataset,labels):

    classList=[example[-1] for example in dataset]

    if classList.count(classList[0]) == len(classList):

        return classList[0]

    if len(dataset[0]) == 1:

        return majorityCnt(classList)

    bestFeat = CART\_chooseBestFeatureToSplit(dataset)

    bestFeatLabel = labels[bestFeat]

    CARTTree = {bestFeatLabel:{}}

    del(labels[bestFeat])

    featValues = [example[bestFeat] for example in dataset]

    uniqueVals = set(featValues)

    for value in uniqueVals:

        subLabels = labels[:]

        CARTTree[bestFeatLabel][value] = CART\_createTree(splitdataset(dataset, bestFeat, value), subLabels)

    return CARTTree

def classify(inputTree, featLabels, testVec):

    firstStr = list(inputTree.keys())[0]

    secondDict = inputTree[firstStr]

    featIndex = featLabels.index(firstStr)

    classLabel = '0'

    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

def classifytest(inputTree, featLabels, testDataSet):

    classLabelAll = []

    for testVec in testDataSet:

        classLabelAll.append(classify(inputTree, featLabels, testVec))

    return classLabelAll

if \_\_name\_\_ == '\_\_main\_\_':

    dataset=[['1','1','1','1','1'],

            ['1','0','1','2','1'],

            ['1','0','1','2','1'],

            ['2','0','1','2','1'],

            ['2','0','1','1','1'],

            ['2','1','0','1','1']]

    print(dataset)

    testSet=[['0','1','1','1'],

            ['2','0','1','2'],

            ['2','0','1','2'],

            ['0','0','1','2'],

            ['2','1','1','1']]

    labels=["年龄","有工作","有自己的房子","信贷情况"]

    labels\_tmp = labels[:]

    CalEnt(dataset)

    ChooseFeatures(dataset)

    CARTdesicionTree = CART\_createTree(dataset,labels\_tmp)

    print('CARTdesicionTree:', CARTdesicionTree)

    print('test result:', classifytest(CARTdesicionTree, labels, testSet))