4.1

假设不存在训练误差为0的决策树，则对于任意学得决策树，至少有一个不为空的节点上存在多个无法划分的数据，则这些数据必然全属于同一类别，与“不含冲突数据”的前提相矛盾。所以假设不成立。即对于不含冲突数据（即特征向量完全相同但标记不同）的训练集，必存在与训练集一致（即训练误差为0）的决策树。

4.3

import pandas as pd

import numpy as np

import json

# 文件读入和处理

df = pd.DataFrame(pd.read\_csv(filepath\_or\_buffer="西瓜数据集4.3.csv", encoding="UTF-8"))

df.drop(labels=["编号"], axis=1, inplace=True)  # 删除编号这一列,inplace=True表示直接在原对象修改

df["好瓜"].replace(to\_replace=["是", "否"], value=["好瓜", "坏瓜"], inplace=True)  # 替换,这一步不是必须的

featureList = df.columns[:-1]  # 除了好瓜属性之外的其他属性

# 离散值属性

featureValue = {}

for feature in featureList[:-2]:

    featureValue[feature] = set(df[feature])  # 各个属性及其出现过的取值,如色泽: (青绿,乌黑,浅白)

# 连续值属性

T = {}  # 候选点

for feature in featureList[-2:]:

    T1 = df[feature].sort\_values()

    T2 = T1[:-1].reset\_index(drop=True)

    T3 = T1[1:].reset\_index(drop=True)

    T[feature] = (T2+T3)/2

# 信息熵

def Ent(D):

    freq = D["好瓜"].value\_counts()/len(D["好瓜"])

    return -sum(pk\*np.log2(pk) for pk in freq)

# 根据离散值划分子集

def split\_discrete(D, feature):

    spiltD = [] #spiltD是由(feature,group)元组所组成的子集列表

    Dgroup = D.groupby(by=feature, axis=0)

    for Dv in Dgroup:

        spiltD.append(Dv)

    return spiltD

#根据连续值划分点划分子集

def split\_continues(D, feature, splitValue):

    spiltD = []

    spiltD.append(D[D[feature] <= splitValue])

    spiltD.append(D[D[feature] > splitValue])

    return spiltD

#计算离散属性的信息增益

def Gain\_discrete(D, feature):

    #Dv是(feature, group)元组,因此Dv[1]才是相应的子集

    gain = Ent(D) - sum(len(Dv[1])/len(D)\*Ent(Dv[1]) for Dv in split\_discrete(D, feature))

    return gain

#计算连续属性的信息增益

def Gain\_continues(D, feature):

    """

    @ return: \_max最大增益 splitValue对应划分点(划分值)

    """

    \_max = 0

    splitValue = 0

    # T[feature]的元素本质上是一个键值对 for t in T[feature]取到的是key而不是value

    # 加.values迭代器取的是对应元素,不加.values的话,取到的是索引值,也就是0123456

    for t in T[feature].values:  # 尝试各个划分点,并取可以使增益最大的划分点

        temp = Ent(D) - sum(len(Dv)/len(D)\*Ent(Dv)

                            for Dv in split\_continues(D, feature, t))

        if \_max < temp:

            \_max = temp

            splitValue = t

    return \_max, splitValue

def chooseBestFeature(D, A):

    informationGain = {}

    for feature in A:

        if feature in ["密度", "含糖率"]:  # 密度和含糖率是连续属性

            ig, splitValue = Gain\_continues(D, feature)

            informationGain[feature+"<=%.3f" % splitValue] = ig

        else:

            informationGain[feature] = Gain\_discrete(D, feature)

    # print(informationGain)

    # informationGain的元素是(feature: 对应的信息增益)键值对,下面是按照信息增益排序的写法

    informationGain = sorted(informationGain.items(),

                             key=lambda ig: ig[1], reverse=True)

    # 返回对应信息增益

    return informationGain[0][0]

def countMajority(D):  # mode()求出现次数最多的元素,iloc取得对应的类：好瓜或坏瓜（是或否）

    # print(D["好瓜"].mode())

    return D["好瓜"].mode().iloc[0]

# print(countMajority(df))

def treeGenerate(D, A):

    # 按照好瓜属性分组,结果只有一个组,也就是 全都是好瓜或全都是坏瓜

    # 表明已经到了叶子节点,返回判断结果

    if len(split\_discrete(D, "好瓜")) == 1:

        return D["好瓜"].iloc[0]

    # 属性集合A为空,或按照A(可能包含不止一个属性)分组只有一个组(所有样本在A上的取值相同),也是到达叶子节点

    # 返回D里数量最多的类型

    if len(A) == 0 or len(split\_discrete(D, A.tolist())) == 1:

        return countMajority(D)

    # 选择信息增益最大的属性

    bestFeature = chooseBestFeature(D, A)

    # print("best feature:", bestFeature)

    if "<=" in bestFeature:  # 连续属性

        bestFeature, splitValue = bestFeature.split("<=")

        myTree = {bestFeature+"<="+splitValue: {}}

        [D0, D1] = split\_continues(D, bestFeature, float(splitValue))

        # 因为A的类别是Index,所以这里直接用Index()复制一份

        # 连续属性在之后的划分还可以继续使用,所以不需要去掉

        A0 = pd.Index(A)

        A1 = pd.Index(A)

        myTree[bestFeature+"<="+splitValue]["yes"] = treeGenerate(D0, A0)

        myTree[bestFeature+"<="+splitValue]["no"] = treeGenerate(D1, A1)

    else:  # discrete

        myTree = {bestFeature: {}}

        for bestFeatureValue, Dv in split\_discrete(D, bestFeature):

            # 在样本集里,bestFeature属性上已经没有bestFeatureValue这一取值

            # 但在实际情况中,是可能还会有bestFeatureValue这一取值的,所以把它们分为D里数量最多的类型

            if len(Dv) == 0:

                return countMajority(D)

            else:

                A2 = pd.Index(A)

                # 离散属性,因为之后的划分不再需要该属性,所以去掉

                A2 = A2.drop([bestFeature])

                Dv = Dv.drop(labels=[bestFeature], axis=1)

                myTree[bestFeature][bestFeatureValue] = treeGenerate(Dv, A2)

    return myTree

if \_\_name\_\_ == "\_\_main\_\_":

    myTree = treeGenerate(df, featureList)

    myTree = json.dumps(myTree, indent=2, ensure\_ascii=False,

                        separators=(',', ':'))  # 这里做个格式化,只为了容易看

    print(myTree)

4.9

