1. （必填）自己提出的问题的理解（罗列全部）：
2. 提出的问题1：P79页，怎么确定a的大小？

讨论后的理解：a的大小应该在实验的过程中通过反复测试调参确定，观察实验结果，确定最合适的取值；

1. 提出的问题2：P80进行剪枝的条件，如果损失函数值变小就剪枝掉，会不会因为样本过少导致分类结果更不准确，有没有其他的剪枝方法或者影响因素？

讨论后的理解：不会,因为CART剪枝采取的剪枝策略是整体最优的，在剪枝时会采用损失函数减小率来进行调整，剪枝幅度更小、产生更多树，选择整体损失函数最小的树进行剪枝；

1. （必填）别人提出的问题的理解（选择几个问题罗列，并给出理解）：
2. 问题3：如何选择分类的特征？

自己的理解：特征选择，就是决策树的构造过程，将特征作为决策树中的节点，特征的划分，目的是保证节点的纯度越来越高，划分样本的能力越强，特征区分效果越好。选择分类特征有三种评判标准，包括信息熵、信息增益和基尼系数，其中信息熵越小，纯度越高，特征划分效果越好；信息增益越大，纯度越高，特征划分效果越好；基尼系数越小，纯度越高，特征划分效果越好；

1. 问题4：如何避免过拟合？

自己的理解：

1. 合理、有效地抽样，用相对能够反映业务逻辑的训练集去产生决策树；
2. 剪枝：提前停止树的增长或者对已经生成的树按照一定的规则进行后剪枝。剪枝是一个简化过拟合决策树的过程，有两种常用的剪枝方法：
   1. 预剪枝： 在决策树生成过程中，对每个结点在划分前先进行估计，若当前结点的划分不能带来决策树泛化性能提升，则停止划分即结束树的构建并将当前节点标记为叶节点；
   2. 后剪枝：首先构造完整的决策树，允许树过度拟合训练数据，然后对那些置信度不够的结点子树用叶子结点来代替，该叶子的类标号用该结点子树中最频繁的类标记；
3. 问题5：cart算法对比id3、C4.5算法有什么优势，为什么要选择cart算法？

自己的理解：

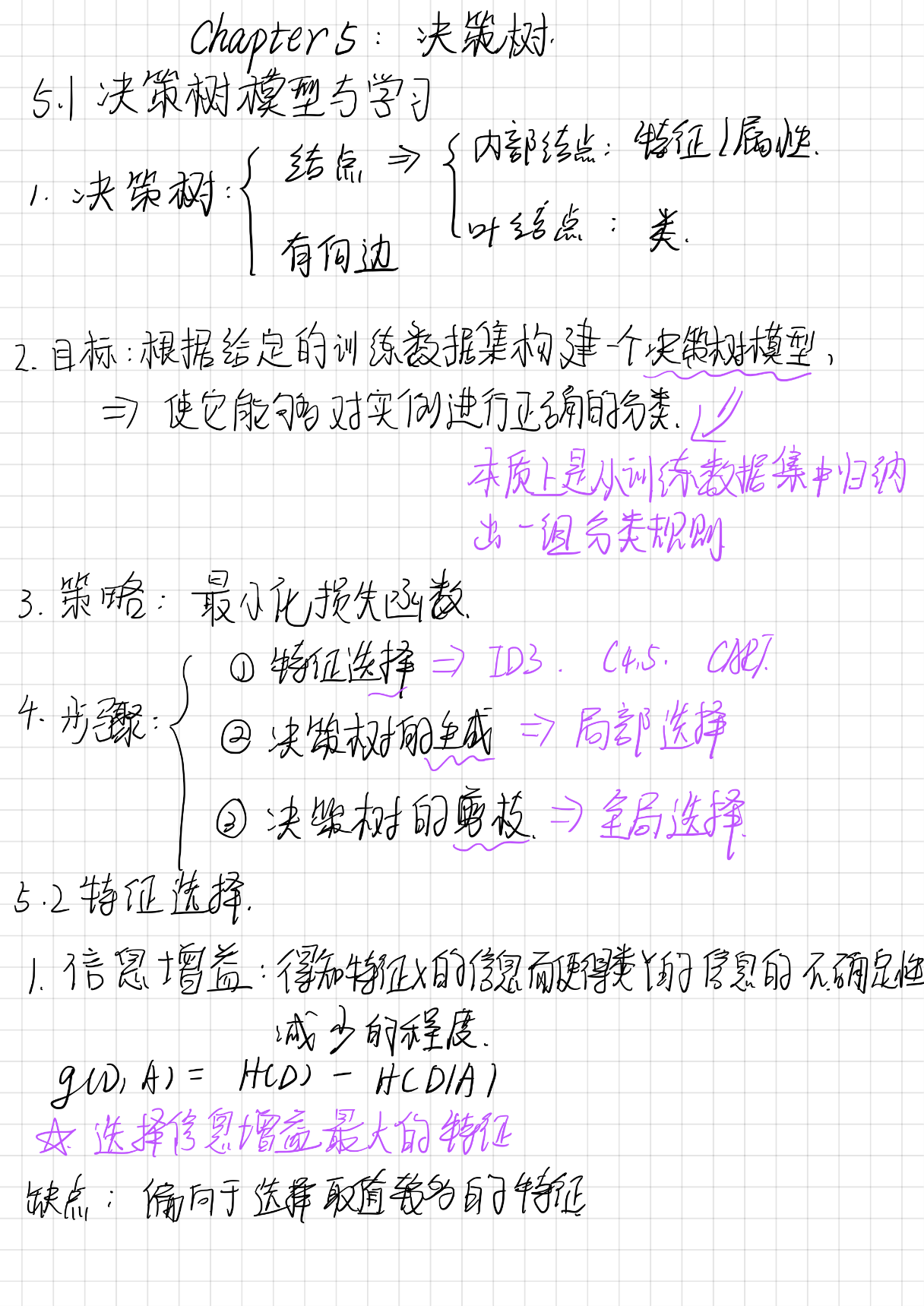
1. cart树是二叉树，不会像多叉树那样形成过多的数据碎片，从而熵会比其他的多分类小一些；
2. 相比ID3和C4.5，CART应用要多一些，既可以用于分类也可以用于回归；
3. （必填）读书计划

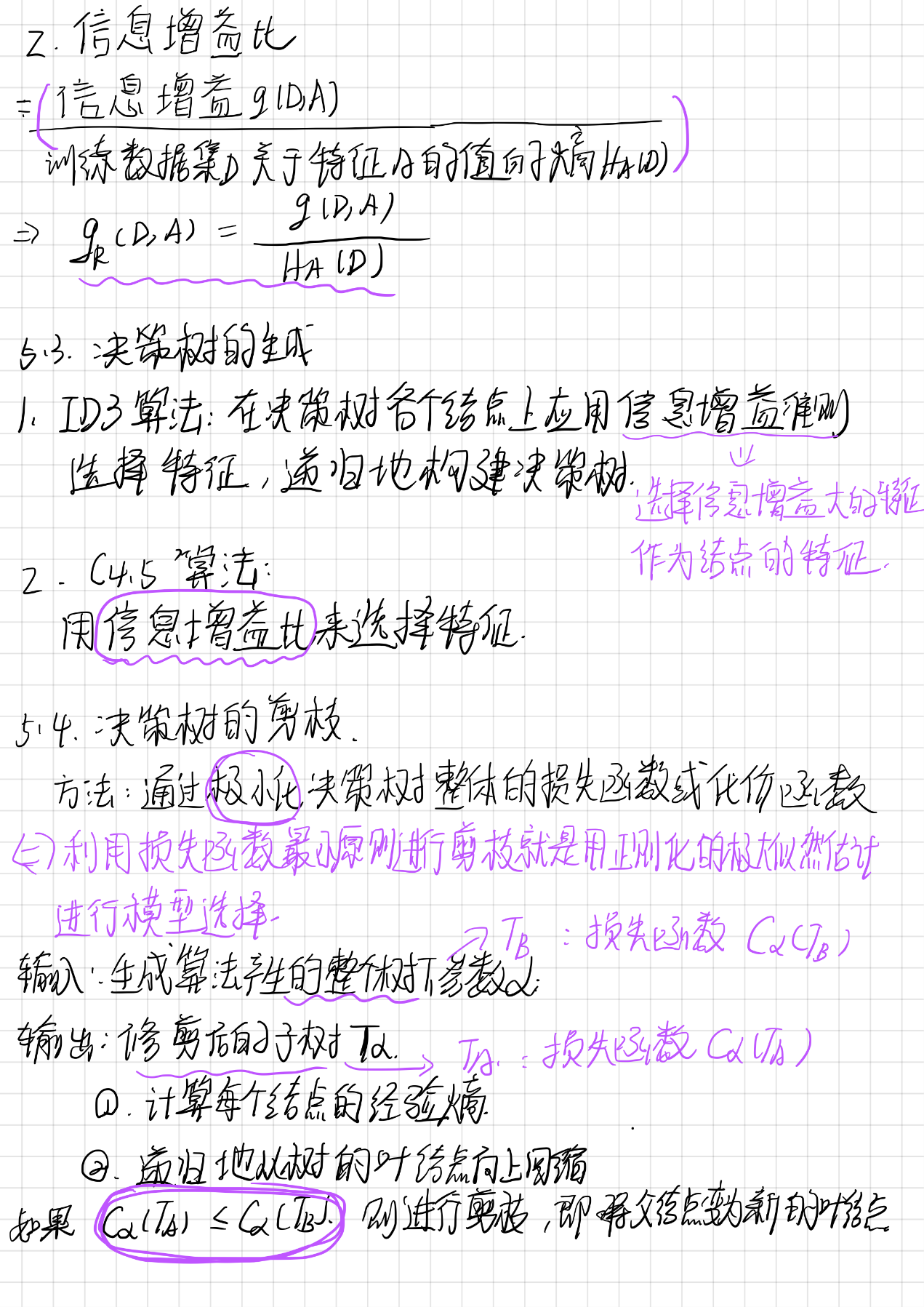
1、本周完成的内容章节：《统计学习方法》第五章

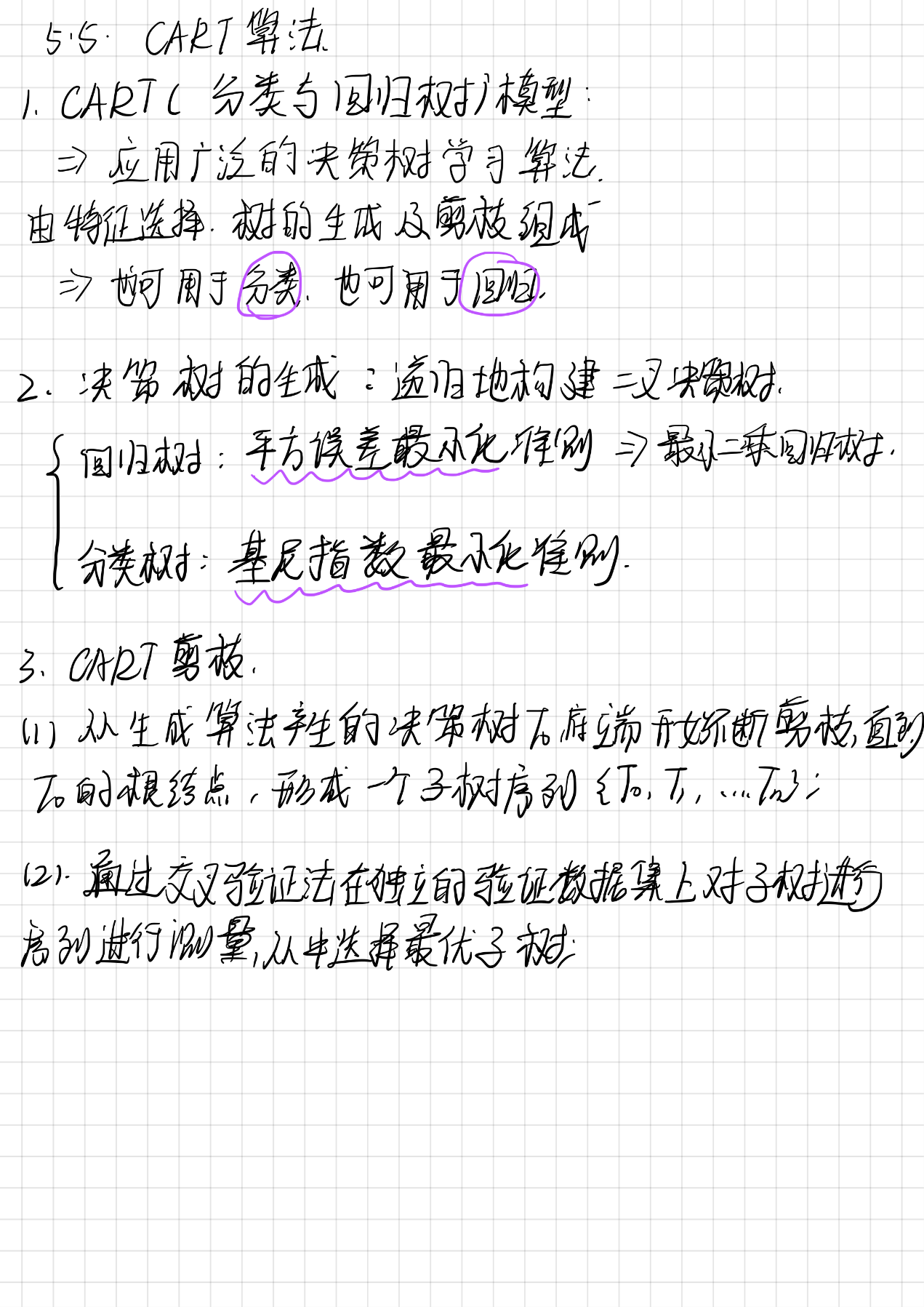
2、下周计划：《统计学习方法》第六章

四、（选做）读书摘要及理解或伪代码的具体实现（读书摘要、伪代码的具体实现代码等可以写到这个部分）

1、读书摘要及理解（选做）：







2、ID3构建决策树伪代码：

from math import log

import operator

def createDataSet1():

# 数据集

dataSet = [('青年', '否', '否', '一般', '不同意'),

('青年', '否', '否', '好', '不同意'),

('青年', '是', '否', '好', '同意'),

('青年', '是', '是', '一般', '同意'),

('青年', '否', '否', '一般', '不同意'),

('中年', '否', '否', '一般', '不同意'),

('中年', '否', '否', '好', '不同意'),

('中年', '是', '是', '好', '同意'),

('中年', '否', '是', '非常好', '同意'),

('中年', '否', '是', '非常好', '同意'),

('老年', '否', '是', '非常好', '同意'),

('老年', '否', '是', '好', '同意'),

('老年', '是', '否', '好', '同意'),

('老年', '是', '否', '非常好', '同意'),

('老年', '否', '否', '一般', '不同意')]

# 特征集

labels = ['年龄', '有工作', '有房子', '信贷情况']

return dataSet,labels

def calcProbabilityEnt(dataSet):

numEntries = len(dataSet) # 数据条数

feaCounts = 0

fea1 = dataSet[0][len(dataSet[0]) - 1]

for featVec in dataSet: # 每行数据

if featVec[-1] == fea1:

feaCounts += 1

probabilityEnt = float(feaCounts) / numEntries

return probabilityEnt

def splitDataSet(dataSet, index, value):

retDataSet = []

for featVec in dataSet:

# 如果该样本该特征的属性值等于传入的属性值，则去掉该属性然后放入数据集中

if featVec[index] == value:

reducedFeatVec = featVec[:index] + featVec[index+1:] # 去掉该属性的当前样本

retDataSet.append(reducedFeatVec) # append向末尾追加一个新元素，新元素在元素中格式不变，如数组作为一个值在元素中存在

return retDataSet

def chooseBestFeatureToSplit(dataSet):

numFeatures = len(dataSet[0]) - 1 # 特征总数

if numFeatures == 1: # 当只有一个特征时

return 0

bestGini = 1 # 最佳基尼系数

bestFeature = -1 # 最优特征

for i in range(numFeatures):

uniqueVals = set(example[i] for example in dataSet) # 去重，每个属性值唯一

feaGini = 0 # 定义特征的值的基尼系数

# 依次计算每个特征的值的熵

for value in uniqueVals:

subDataSet = splitDataSet(dataSet,i,value) # 根据该特征属性值分的类

# 参数：原数据、循环次数(当前属性值所在列)、当前属性值

prob = len(subDataSet) / float(len(dataSet))

probabilityEnt = calcProbabilityEnt(subDataSet)

feaGini += prob \* (2 \* probabilityEnt \* (1 - probabilityEnt))

if (feaGini < bestGini): # 基尼系数越小越好

bestGini = feaGini

bestFeature = i

return bestFeature

def majorityCnt(classList):

classCount = {}

# 计算每个类别出现次数

for vote in classList:

try:

classCount[vote] += 1

except KeyError:

classCount[vote] = 1

sortedClassCount = sorted(classCount.items(),key = operator.itemgetter(1),reverse = True) # 出现次数最多的类别在首位

# 对第1个参数，按照参数的第1个域来进行排序（第2个参数），然后反序（第3个参数）

return sortedClassCount[0][0] # 该属性的类别

def createTree(dataSet,labels):

classList = [example[-1] for example in dataSet] # 获取每行数据的最后一个值，即每行数据的类别

# 当数据集只有一个类别

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

return classList[0]

# 当数据集只剩一列（即类别），即根据最后一个特征分类

if len(dataSet[0]) == 1:

return majorityCnt(classList)

# 其他情况

bestFeat = chooseBestFeatureToSplit(dataSet) # 选择最优特征（所在列）

bestFeatLabel = labels[bestFeat] # 最优特征

del(labels[bestFeat]) # 从特征集中删除当前最优特征

uniqueVals = set(example[bestFeat] for example in dataSet) # 选出最优特征对应属性的唯一值

myTree = {bestFeatLabel:{}} # 分类结果以字典形式保存

for value in uniqueVals:

subLabels = labels[:] # 深拷贝，拷贝后的值与原值无关（普通复制为浅拷贝，对原值或拷贝后的值的改变互相影响）

myTree[bestFeatLabel][value] = createTree(splitDataSet(dataSet,bestFeat,value),subLabels) # 递归调用创建决策树

return myTree

if \_\_name\_\_ == '\_\_main\_\_':

dataSet, labels = createDataSet1() # 创造示列数据

print(createTree(dataSet, labels)) # 输出决策树模型结果