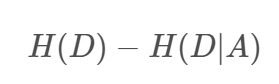
1. （必填）自己提出的问题的理解（罗列全部）：
2. **提出的问题1：**

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：

考虑一个极端情况，某个属性（特征）的取值很多，以至于每一个取值对应的类别只有一个。这样根据



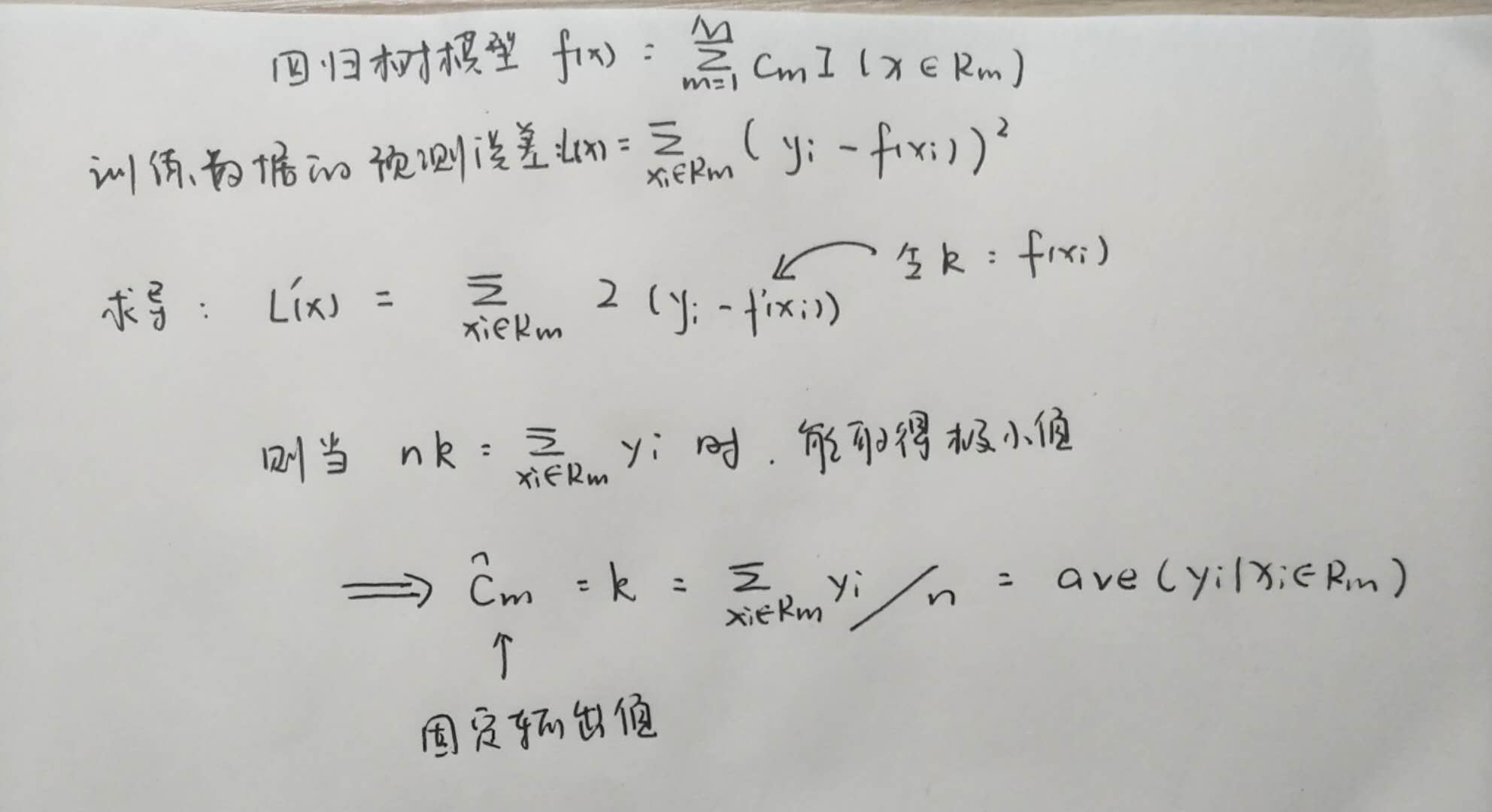
可以得知后面的那一项的值为0，因此得到的信息增益就会越大。所以当算法以信息增益作为划分训练集的特征时，就会偏向于选择值更多的特征。

1. **提出的问题2：**

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：

这是一个通过求导来求极值的问题

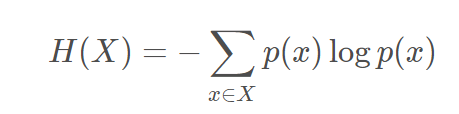


1. （必填）别人提出的问题的理解（选择几个问题罗列，并给出理解）：
2. **问题3**：

为什么说熵只和分布有关，而和值无关？

自己的理解：

根据熵的计算公式



可以知道熵只与p(x)有关，p(x)即为X分布的概率。因此，熵只与分布有关，而与x的值无关。

1. **问题4：**

cart算法对比id4.5算法有什么优势，为什么要选择cart算法？

自己的理解：

C4.5的缺点：

* + - 剪枝策略可以再优化；
    - C4.5 用的是多叉树，用二叉树效率更高；
    - C4.5 只能用于分类；
    - C4.5 使用的熵模型拥有大量耗时的对数运算，连续值还有排序运算；
    - C4.5 在构造树的过程中，对数值属性值需要按照其大小进行排序，从中选择一个分割点，所以只适合于能够驻留于内存的数据集，当训练集大得无法在内存容纳时，程序无法运行。

CART的优点：

* + - 可以生成可以理解的规则
    - 计算量相对来说不是很大
    - 可以处理连续和种类字段
    - 决策树可以清晰的显示哪些字段比较重要

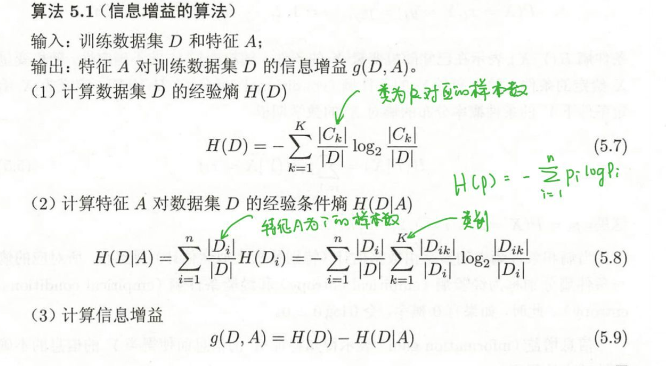
1. **问题5：**

如何选择分类的特征

讨论后的理解：

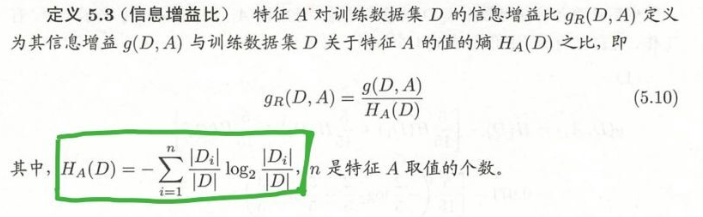
有两种方案来选择分类特征

方案一：选择信息增益最大的特征，信息增益计算方法如下：



但是以信息增益作为选择特征的方案，会存在更倾向于选择值更多的特征的问题，因此提出了以信息增益比作为衡量标准的方案。

方案二：选择信息增益比最大的特征



三、（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：决策树）

2、下周计划：统计学习方法（第六章：逻辑斯蒂回归与最大熵模型）

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）

本章主要介绍了决策树的基本概念和模型，以及几个典型的算法，包括ID3、C4.5和CART算法。其中，决策树学习算法包括特征选择、决策树的生成和决策树的剪枝，因此本章以此为线索介绍了这几个重要部分。

**特征选择：**

特征选择主要包括两种方案，信息增益和信息增益比，其中信息增益是指选择某一特征后能降低的熵的大小，因此是熵减去条件熵的差值。但是根据这一定义可以发现，这一个方案会更倾向于选择值更多的特征，为了避免了这一问题，提出了新的方案：信息增益比。信息增益比是指信息增益与训练集关于特征的熵之比。

**决策树的生成：**

决策树生成学习局部的模型

ID3算法：相当于用极大似然法进行概率模型的选择。

ID3算法是J. Ross Quinlan于1975提出的一种贪心算法，用来构造决策树。其建立在“奥卡姆剃刀”的基础上，即越是小型的决策树越优于大的决策树。ID3算法中根据特征选择和信息增益评估，每次选择信息增益最大的特征作为分支标准。ID3算法可用于划分标称型数据集，没有剪枝的过程，为了去除过度数据匹配的问题，可通过裁剪合并相邻的无法产生大量信息增益的叶子节点（例如设置信息增益阀值）。使用信息增益其实是有一个缺点，那就是它偏向于具有大量值的属性–就是说在训练集中，某个属性所取的不同值的个数越多，那么越有可能拿它来作为分裂属性，而这样做有时候是没有意义的。此外，ID3不能处理连续分布的数据特征。

C4.5算法：

C4.5算法是对ID3算法的改进，C4.5克服了ID3的2个缺点：

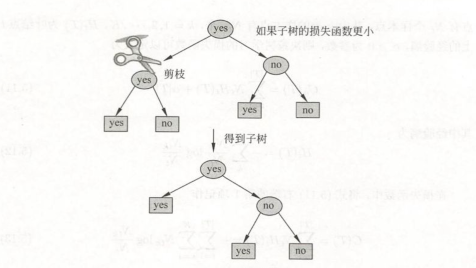
* + 用信息增益选择属性时偏向于选择分枝比较多的属性值，即取值多的属性不能处理连续属性
  + 对于离散特征，C4.5算法不直接使用信息增益，而是使用“增益率”（gain ratio）来选择最优的分支标准

**决策树的剪枝：**

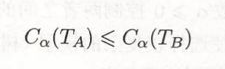
决策树剪枝学习整体的模型

（1）计算每个节点的经验熵

（2）递归地从树的叶节点向上收缩



并且满足



时，进行剪枝。

（3）返回（2），知道不能继续位置，得到损失函数最小的子树T

2、伪代码的具体实现(选做)

ID3算法的实现：

import math

import operator

def calcShannonEnt(dataset):

numEntries = len(dataset)

labelCounts = {}

for featVec in dataset:

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])/numEntries

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

return shannonEnt

def CreateDataSet():

dataset = [[1, 1, 'yes' ],

[1, 1, 'yes' ],

[1, 0, 'no'],

[0, 1, 'no'],

[0, 1, 'no']]

labels = ['no surfacing', 'flippers']

return dataset, labels

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 chooseBestFeatureToSplit(dataSet):

numberFeatures = len(dataSet[0])-1

baseEntropy = calcShannonEnt(dataSet)

bestInfoGain = 0.0;

bestFeature = -1;

for i in range(numberFeatures):

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

uniqueVals = set(featList)

newEntropy =0.0

for value in uniqueVals:

subDataSet = splitDataSet(dataSet, i, value)

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

newEntropy += prob \* calcShannonEnt(subDataSet)

infoGain = baseEntropy - newEntropy

if(infoGain > bestInfoGain):

bestInfoGain = infoGain

bestFeature = i

return bestFeature

def majorityCnt(classList):

classCount ={}

for vote in classList:

if vote not in classCount.keys():

classCount[vote]=0

classCount[vote]=1

sortedClassCount = sorted(classCount.iteritems(), key=operator.itemgetter(1), reverse=True)

return sortedClassCount[0][0]

def 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 = chooseBestFeatureToSplit(dataSet)

bestFeatLabel = labels[bestFeat]

myTree = {bestFeatLabel:{}}

del(labels[bestFeat])

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

uniqueVals = set(featValues)

for value in uniqueVals:

subLabels = labels[:]

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

return myTree

myDat,labels = CreateDataSet()

createTree(myDat,labels)