**基于决策树的贷款申请项目**

**决策树（decision tree）**是一种基本的分类与回归方法。在分类问题中，表示基于特征对实例进行分类的过程。主要优点是模型具有可读性，分类速度快。

决策树包含3个步骤：**特征选择、决策树的生成、决策树的修剪**。

决策树学习通过**递归选择最优特征**，并根据该特征对训练数据进行分割，使得对各个子数据集有一个最好的分类的过程。这一过程对应着对特征空间的划分，也对应着决策树的构建。首先，构建根结点，将所有训练数据都放在根结点。选择一个最优特征，按照这一特征将训练数据集分割成子集，使得各个子集有一个在当前条件下最好的分类。如果这些子集已经能够被基本正确分类，那么构建叶结点，并将这些子集分到所对应的叶结点中去；如果还有子集不能被基本正确分类，则对这些子集选择新的最优特征，继续对其进行分割，构建相应的节点。如此递归地进行下去，直至所有训练数据子集被基本正确分类，或者没有合适的特征为止。最后每个子集都被分到叶结点上，即都有了明确的类，即生成了一棵决策树。

决策树主要算法有：**ID3、C4.5、CART**，进化后的C4.5算法C5.0、分类有极大提升的Tsallis算法。这些算法的区别就在于**选择最优特征**的方式。C5.0的核心原理与C4.5是相同，C4.5的速度非常慢，而C5.0对大数据运算效率极高。其中，ID3算法是决策树的**基础算法**。

**一、基于决策树的贷款申请项目**

下表示由15个样本组成的贷款申请训练数据。数据包括贷款申请人的4个特征（属性）：年龄（青年、中年、老年）、工作（是，否）、有自己的房子（是、否）、信贷情况（非常好、好、一般）。表的最后一列是类别，是否同意贷款：是、否。

**贷款申请样本数据表**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | **年龄** | **有工作** | **有自己的房子** | **信贷情况** | **类别：同意贷款** |
| 1 | 青年 | 是 | 是 | 一般 | 是 |
| 2 | 青年 | 是 | 否 | 好 | 是 |
| 3 | 青年 | 否 | 否 | 好 | 否 |
| 4 | 青年 | 否 | 否 | 一般 | 否 |
| 5 | 青年 | 否 | 否 | 一般 | 否 |
| 6 | 中年 | 是 | 是 | 好 | 是 |
| 7 | 中年 | 否 | 是 | 非常好 | 是 |
| 8 | 中年 | 否 | 是 | 非常好 | 是 |
| 9 | 中年 | 否 | 否 | 好 | 否 |
| 10 | 中年 | 否 | 否 | 一般 | 否 |
| 11 | 老年 | 是 | 否 | 非常好 | 是 |
| 12 | 老年 | 是 | 否 | 好 | 是 |
| 13 | 老年 | 否 | 是 | 非常好 | 是 |
| 14 | 老年 | 否 | 是 | 好 | 是 |
| 15 | 老年 | 否 | 否 | 一般 | 否 |

本项目通过所给的训练数据学习一个贷款申请的决策树，用以对未来的贷款申请进行分类，即当新的客户提出贷款申请时，根据申请人的特征，利用决策树决定是否批准贷款申请。

ID3算法是通过信息增益来选择最优特征的。

**1.熵与互信息**

熵是表示随机变量不确定性的度量。则信息熵代表信息的不确定性，信息的不确定性越大，熵越大。例如，“太阳从东方升起”这句话代表的信息可以认为为0，因为太阳从东方升起是一个特定的规律。因此，信息熵和事件发生的概率成反比。信息熵又称为香农熵。

互信息是指两个随机变量之间的关联程度。即给定一个随机变量后，另一个随机变量不确定性的削弱程度。因而互信息取值最小为0，这意味着给定一个随机变量对确定另一个随机变量没有关系。最大取值为随机变量的熵，意味着给定一个随机变量能完全消除另一个随机变量的不确定性。

以下列出公式：

设X是一个取有限个值的离散随机变量，其概率分布为：

IMG_256 IMG_256

则随机变量X的熵定义为：

根据上面的定义可知，熵只依赖于X的分布，而与X的取值无关，因此，可将X的熵记为H(p)，则

条件熵IMG_256表示在已知随机变量X的条件下随机变量Y的不确定性：

 IMG_256 IMG_256

当熵和条件熵的概率由数据估计（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。

信息增益（information gain）表示得知特征X的信息而使得类Y的信息不确定性减少的程度。特征A对训练数据集D的信息增益g(D,A)定义为集合D的经验熵H(D)与特征A给定条件下D的经验条件熵H(D|A)之差，即：

IMG_256

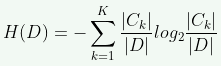
一般地，熵H(Y)与条件熵H(Y|X)之差称为互信息。则此处决策树学习中的信息增益等价于训练数据集中类与特征的互信息。

根据上面的定义可以看到，信息增益就表示由于特征A而使得对数据集D的分类的不确定性减少的程度。显然，信息增益大的特征具有更强的分类能力。

**2.信息增益的算法**

输入：训练数据集D和特征A；输出：特征A对训练数据集D的信息增益g(D,A)

（1）计算数据集D的经验熵H(D)

即类Ck在数据集D中的概率。其中，D为训练数据集，|D|为样本容量（即个数），Ck为第K个类，|Ck|为第K个类容量（即个数）,则

（2）计算特征A对数据集D的经验条件熵H(D|A)

其中，Di为根据特征A划分的子集，|Di|为样本容量（即个数），Dik为Di属于Ck的样本的集合，|Dik|为Dik的样本的个数，则 

（3）计算信息增益 IMG_256

**3.计算信息增益**

首先计算数据本身的信息熵H(D)：

一共有15个样本，单看类别，有9个是，6个否，对于原公式

K为2，C1为是，C2为否，因此

然后计算各特征对数据集D的信息增益。分别以A1、A2、A3、A4表示年龄、有工作、有自己的房子和信贷情况4个特征，则



（1）对于年龄，

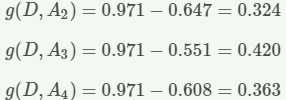
Di有三种情况：青年、中年、老年，它们分别有5个样本，即分别占15个中的5个；而Dik，k为类别的个数，即是、否；

则D11为青年中同意贷款（类别：是）的样本，即5个中有2个，D12为青年中不同意贷款（类别：否）的样本，即5个中有3个；中年和老年类似。



因此

（2）类似地，求出有工作、有自己的房子、信贷情况的信息增益，分别为：



（3）比较四个信息增益值，**A3（有自己的房子）对应的信息增益值最大**，因此选择特征**A3作为最优特征**。

**4.ID3算法**

ID3算法的核心是在决策树各个节点上**应用信息增益准则**选择特征。

输入：训练数据集D，特征集A，阈值u（可选）；输出：决策树T

（1）判断D中所有样本的类别，如果属于同一类，则T为单结点树，并且该类为类标记，返回T。例如：此时样本为如下表格所示，共3个样本，虽然每个样本的同一个特征的属性值都不相同，但它们都属于同一类：M。则此时决策树T为单节点树，即只有根结点M的树。

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **特征1** | **特征2** | **类别** |
| 1 | A | D | M |
| 2 | B | E | M |
| 3 | C | F | M |

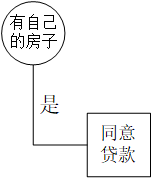
如果特征集A为空集，则T为单结点树，将D中样本数最大的类作为该结例如，此时样本为如下表格，共3个样本，由于特征集A为空，因此只需要看类别，此时三个样本中2个样本类别为M，1个样本类别为N，则M类的样本数量最多，因此认为该结点对应的类别为M，即样本123的类别都分类M。那么，如果此时出现一个未知数据，特征1为B，特征2为E，根据决策树来判断该数据类别，则类别为M。这种情况下会出现误判，因为它和样本2的特征属性完全一致，则类别应该为N。

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **特征1** | **特征2** | **类别** |
| 1 | A | D | M |
| 2 | B | E | N |
| 3 | C | F | M |

（3）否则，根据上面的计算方法计算特征集A中每个特征对D的信息增益，选择信息增益最大的特征Ag。

（4）如果Ag的信息增益小于阈值u，则T为单结点树，将D中样本数最大的类别作为该结点的类标记，返回T。例如，上面例子中，如果设定阈值为0.45，则没有特征的信息增益大于它（最大的有自己的房子对应的信息增益值为0.42）。因此找到上面15个样本中最大的类别，即同意贷款作为该结点的类标记，这15个样本都属于该类。这种情况下同样会出现误判。例如，此时出现一个未知类别的数据，它的特征属性与第15个样本一致，根据决策树来判断它的类别，则为同意贷款，但实际上第15个样本的类别为否，即不同意贷款，那么它也应该是不同意贷款。

（5）否则，对Ag的每一个可能值ai，依Ag=ai将D分割为若干非空子集Di，将Di中样本数最大的类作为标记，构建子结点，由结点及子结点构成树T，返回T。

（6）对第i个子结点，以Di为训练集，以A-｛Ag｝为特征集，递归地调用第1-5步，得到子树Ti，返回Ti。例如，如果没有设定阈值，那么，已经得到了A3（有自己的房子）作为第一个特征，即A3为根结点，根据A3=是，A3=否，将之分为两类：

A3=是的样本，它的类别全都为是，此时满足了第1步所指的，全都为同一类，则直接将类别是作为子结点，构建了下面这样的树：

A3=否的样本，它的类别既有是，又有否，因此继续递归地调用第1-5步，此时的样本集为有自己的房子对应的属性值为否的剩余样本，特征集为去掉有自己的房子这个特征的剩余特征。

**5.决策树的优化**

上面的ID3算法中提到了输入的阈值，阈值可以有很多方式，它是结束决策树的条件。一般情况下，阈值与特征因子进行比较，例如第4步提到的。但是， 有时候，也会有其他的选择方式，例如设定树的最大深度等。也可以不设阈值，则此时决策树的结束条件是特征集为空集。

上面的ID3算法中还提到了可能对类别进行误判，一般这种情况只可能出现在最后一次建立子树时。

一般情况下，决策树产生的树往往对训练数据的分类很准确，尤其是不设置阈值时。但当用于对未知数据分类时却没有那么准确，即出现过拟合现象。因为当分类过于细致，当一个属性不对应，那么就很可能出现不能分类的情况。

解决建立决策树时易发生**过拟合现象的方法：剪枝、随机森林**。

剪枝即是从已生成的树上裁减掉一些子树或者叶结点，并将其根结点或父结点作为新的叶结点。

随机森林是**根据样本集和特征集，建立很多的决策树**，每一棵树对于同一个未知数据都会有一个类别，统计这些类别，**类别数最多的那个类即为未知数据的类别**。

**二、决策树之C4.5生成算法**

**1.ID3算法的缺陷**

对于ID3算法，它存在以下问题：

取值多的属性，更容易使数据更纯，其信息增益更大；

训练得到的是一棵庞大且深度浅的树，这样的树是不合理的。

而C4.5算法可以抑制ID3的上述缺点。

**2.C4.5算法**

C4.5算法改进了信息增益，它选用信息增益比来选择最优特征。

特征A对训练数据集D的信息增益比gR(D,A)定义为其信息增益g(D,A)与训练数据集D关于特征A的值的熵HA(D)之比，即





n是特征A取值的个数。

其他部分与ID3算法完全相同。

**三、ID3和C4.5算法的Python实现**

**1.算法流程**

**（1）决策树流程：**

获取训练集D和特征集A

传入数据集D：计算D的经验熵

计算个特征对数据集的信息增益（比）

比较信息增益，选出最大值对应的特征

根据特征将数据集D划分为D1、D2、...、Dn

判断Di是否为一类：

若是，则为一类

若非，计算Di经验熵

**（2）信息增益（比）**

循环求和：

获取属性B1所在行及长度：计算特征B的属性B1占所有数据比

计算数据集属性B1的信息熵

计算这二者的积，或计算每个A1占比\*log2A1占比之差（即特征的值的熵）

特征B对数据集的信息增益=原数据集-和，或信息增益比=信息增益/特征的值的熵

对上面循环求解并比较每个特征对数据集的信息增益（比），求出最大值

**2.C4.5算法的Python实现**

from math import log

import operator

def createDataSet1():

"""

创造示例数据/读取数据

@param dataSet: 数据集

@return dataSet labels：数据集 特征集

"""

# 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 calcShannonEnt(dataSet):

"""

计算数据的熵(entropy)

@param dataSet: 数据集

@return shannonEnt: 数据集的熵

"""

numEntries = len(dataSet) # 数据条数

# 循环判断每个样本的类别，统计每个类别的样本总数

labelCounts = {}

for featVec in dataSet:

currentLabel = featVec[-1] # 当前样本类型

# 统计每个样本类型的数量

try:

labelCounts[currentLabel] += 1

except KeyError:

labelCounts[currentLabel] = 1

# 根据公式计算香浓熵

shannonEnt = 0

for key in labelCounts:

prob = float(labelCounts[key]) / numEntries

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

return shannonEnt

def splitDataSet(dataSet, index, value):

"""

划分数据集，提取含有某个特征的某个属性的所有数据

@param dataSet: 数据集

@param index: 属性值所对应的特征列

@param value: 某个属性值

@return retDataSet: 含有某个特征的某个属性的数据集

"""

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):

"""

选择最优特征

@param dataSet: 数据集

@return bestFeature: 最优特征所在列

"""

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

if numFeatures == 1: # 当只有一个特征时

return 0

baseEntropy = calcShannonEnt(dataSet) # 数据集的熵

bestInfoGainRatio = 0 # 最佳信息增益比

bestFeature = -1 # 最优特征所在列

for i in range(numFeatures): # range(5) 代表从0到5，不包括5

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

newEntropy = 0 # 定义按特征分类后的熵

feaEntropy = 0 # 定义特征的值的熵

# 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))

newEntropy += prob \* calcShannonEnt(subDataSet)

feaEntropy -= prob \* log(prob, 2)

infoGainRatio = (baseEntropy - newEntropy) / feaEntropy # 信息增益比

if (infoGainRatio > bestInfoGainRatio):

bestInfoGainRatio = infoGainRatio

bestFeature = i

return bestFeature

def majorityCnt(classList):

"""

对最后一个特征分类，出现次数最多的类即为该属性类别，比如：最后分类为2男1女，则判定为男

@param classList: 数据集，也是类别集

@return sortedClassCount[0][0]: 该属性的类别

"""

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):

"""

对最后一个特征分类，按分类后类别数量排序，比如：最后分类为2同意1不同意，则判定为同意

@param dataSet: 数据集

@param labels: 特征集

@return myTree: 决策树

"""

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

**3.ID3算法的Python实现**

根据上面的分析，将chooseBestFeatureToSplit函数中信息增益比的求解修改为信息增益的求解即可，即修改第92行：

infoGainRatio= baseEntropy - newEntropy # 信息增益 修改第92行

由于infoGainRatio为信息增益比，infoGain才为信息增益，因此为了使变量名正确表示其含义，修改chooseBestFeatureToSplit函数的相关变量名，最后函数为：def chooseBestFeatureToSplit(dataSet):

"""

选择最优特征

@param dataSet: 数据集

@return bestFeature: 最优特征所在列

"""

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

if numFeatures == 1: # 当只有一个特征时

return 0

baseEntropy = calcShannonEnt(dataSet) # 数据集的熵

bestInfoGain = 0 # 最佳信息增益比

bestFeature = -1 # 最优特征所在列

for i in range(numFeatures): # range(5) 代表从0到5，不包括5

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

newEntropy = 0 # 定义按特征分类后的熵

feaEntropy = 0 # 定义特征的值的熵

# 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))

newEntropy += prob \* calcShannonEnt(subDataSet)

feaEntropy -= prob \* log(prob, 2)

infoGain = baseEntropy - newEntropy # 信息增益比

if (infoGain > bestInfoGain):

bestInfoGain = infoGain

bestFeature = i

return bestFeature