# 实验三 决策树

**一、实验目的**

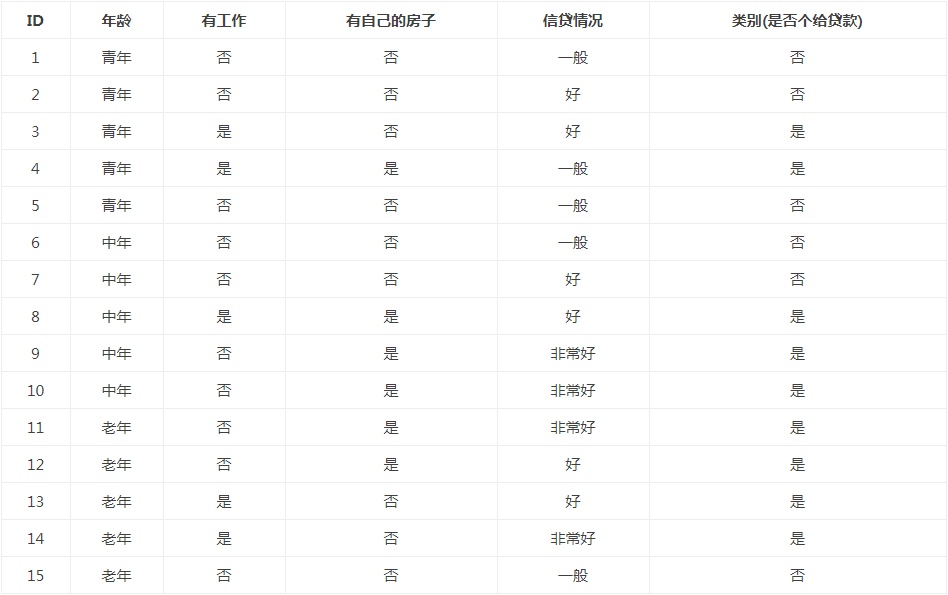
通过本实验熟悉决策树的算法流程，并能够使用决策树预测隐形眼镜的类型。

1. **实验内容**

本实验希望通过所给的训练数据学习一个贷款申请的决策树，用于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。通过该实验来熟悉决策树的算法流程，进一步学习使用sklearn中的决策树模型来预测隐形眼镜的类型。

1. **数据准备**

首先让我们看一组实例，贷款申请样本数据表。从表中可以看到有年龄、是否有工作、是否有自己的房子、信贷情况等属性。



我们希望通过该组数据训练学习一个贷款申请决策树，用于对未来的贷款申请进行分类。

在编写代码之前，首先要对数据集进行属性标注。（年龄：0代表青年，1代表中年，2代表老年；有工作：0代表否，1代表是；有自己的房子：0代表否，1代表是；信贷情况：0代表一般，1代表好，2代表非常好；类别(是否给贷款)：no代表否，yes代表是）。确定好这些，就可以创建数据集了，函数createDataSet用于创建数据集，返回数据集和属性标签：

# -\*- coding: UTF-8 -\*-

from math import log

def createDataSet():

dataSet = [[0, 0, 0, 0, 'no'], #数据集

[0, 0, 0, 1, 'no'],

[0, 1, 0, 1, 'yes'],

[0, 1, 1, 0, 'yes'],

[0, 0, 0, 0, 'no'],

[1, 0, 0, 0, 'no'],

[1, 0, 0, 1, 'no'],

[1, 1, 1, 1, 'yes'],

[1, 0, 1, 2, 'yes'],

[1, 0, 1, 2, 'yes'],

[2, 0, 1, 2, 'yes'],

[2, 0, 1, 1, 'yes'],

[2, 1, 0, 1, 'yes'],

[2, 1, 0, 2, 'yes'],

[2, 0, 0, 0, 'no']]

labels = ['年龄', '有工作', '有自己的房子', '信贷情况'] #分类属性

return dataSet, labels #返回数据集和分类属性

1. **特征选择**

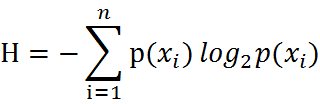
特征选择在于选取对训练数据具有分类能力的特征。这样可以提高决策树学习的效率，如果利用一个特征进行分类的结果与随机分类的结果没有很大差别，则称这个特征是没有分类能力的。经验上扔掉这样的特征对决策树学习的精度影响不大。通常特征选择的标准是信息增益。

* 1. **香农熵**

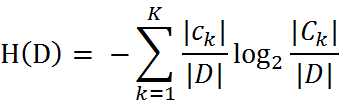
在计算信息增益前，首先要计算香农熵（熵）。熵定义为信息的期望值。在信息论与概率统计中，熵是表示随机变量不确定性的度量。如果待分类的事物可能划分在多个分类之中，则符号xi的信息定义如下，其中p(xi)是选择该分类的概率：

机器学习实战教程（二）：决策树基础篇之让我们从相亲说起

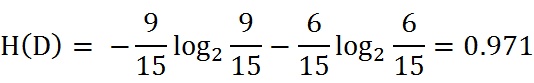
通过上式，我们可以得到所有类别的信息。为了计算熵，我们需要计算所有类别所有可能值包含的信息期望值(数学期望)，通过下面的公式得到：（其中n是分类的数目。熵越大，随机变量的不确定性就越大。）



当熵中的概率由数据估计(特别是最大似然估计)得到时，所对应的熵称为经验熵。我们定义贷款申请样本数据表中的数据为训练数据集D，则训练数据集D的经验熵为H(D)，|D|表示其样本容量，及样本个数。设有K个类Ck, = 1,2,3,...,K,|Ck|为属于类Ck的样本个数，因此经验熵公式就可以写为：



根据此公式计算经验熵H(D)，分析上节中贷款申请样本数据表中的数据。最终分类结果只有两类，即放贷和不放贷。根据表中的数据统计可知，在15个数据中，9个数据的结果为放贷，6个数据的结果为不放贷。所以数据集D的经验熵H(D)为：



下面编写函数calcShannonEnt来计算经验熵。前面关于标签统计代码已经实现，你的任务是实现经验熵的计算部分，首先计算选择每个标签的概率，再用公式计算经验熵并返回。运行代码，打印训练数据集和经验熵（HD），如果结果和我们计算的结果一致，则代码实现是正确的。

def calcShannonEnt(dataSet):

numEntires = len(dataSet)

labelCounts = {} #保存每个标签(Label)出现次数的字典

for featVec in dataSet:

currentLabel = featVec[-1] #提取标签(Label)信息

if currentLabel not in labelCounts.keys():

labelCounts[currentLabel] = 0

labelCounts[currentLabel] += 1

shannonEnt = h(labelCounts.values())

# print(list(labelCounts.values()),'--',sum(list(labelCounts.values())))

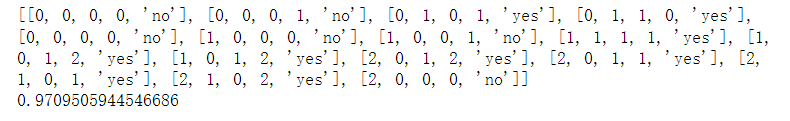
return shannonEnt

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

dataSet, features = createDataSet()

print(dataSet)

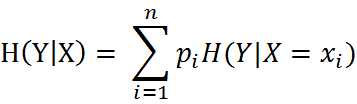
print(calcShannonEnt(dataSet))



* 1. **信息增益**

信息增益是相对于特征而言的，信息增益越大，特征对最终的分类结果影响也就越大，我们就应该选择对最终分类结果影响最大的那个特征作为我们的分类特征。计算信息增益前还需明确条件熵。

条件熵H(Y|X)表示在已知随机变量X的条件下随机变量Y的不确定性，随机变量X给定的条件下随机变量Y的条件熵(conditional entropy)H(Y|X)，定义为X给定条件下Y的条件概率分布的熵对X的数学期望：



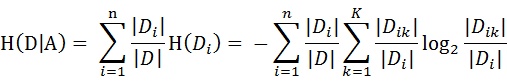
这里，

机器学习实战教程（二）：决策树基础篇之让我们从相亲说起

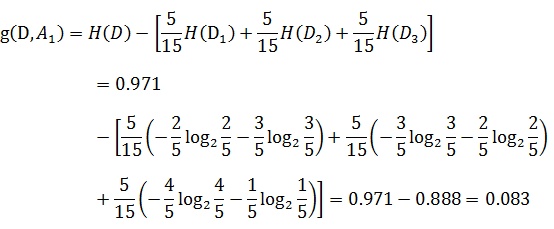
同理，当条件熵中的概率由数据估计(特别是极大似然估计)得到时，所对应的条件熵称为条件经验熵。特征A对训练数据集D的信息增益g(D,A)，定义为集合D的经验熵H(D)与特征A给定条件下D的经验条件熵H(D|A)之差，即：

机器学习实战教程（二）：决策树基础篇之让我们从相亲说起

设特征A有n个不同的取值{a1,a2,···,an}，根据特征A的取值将D划分为n个子集{D1,D2，···,Dn}，|Di|为Di的样本个数。记子集Di中属于Ck的样本的集合为Dik，即Dik = Di ∩ Ck，|Dik|为Dik的样本个数。于是经验条件熵的公式可以写为：



以贷款申请样本数据表为例进行说明。看下年龄这一列的数据，也就是特征A1，一共有三个类别，分别是：青年、中年和老年。我们只看年龄是青年的数据，年龄是青年的数据一共有5个，所以年龄是青年的数据在训练数据集出现的概率是十五分之五，也就是三分之一。同理，年龄是中年和老年的数据在训练数据集出现的概率也都是三分之一。现在我们只看年龄是青年的数据的最终得到贷款的概率为五分之二，因为在五个数据中，只有两个数据显示拿到了最终的贷款，同理，年龄是中年和老年的数据最终得到贷款的概率分别为五分之三、五分之四。所以计算年龄的信息增益，过程如下：



同理，其余特征的信息增益g(D,A2)、g(D,A3)和g(D,A4)。分别为0.324、0.420、0.363，最后，比较特征的信息增益，由于特征A3(有自己的房子)的信息增益值最大，所以选择A3作为最优特征。

下面我们将编写代码。splitDataSet函数是用来选择各个特征子集的，比如调用splitDataSet(dataSet,0,0)这样返回的子集就是年龄为青年的5个数据集，参数dataSet为带划分的数据集，axis为划分数据集的特征，value表示需要返回的特征的值：

def splitDataSet(dataSet, axis, value):

retDataSet = [] #创建返回的数据集列表

for featVec in dataSet:

if featVec[axis] == value:

reducedFeatVec = featVec[:axis] #去掉axis特征

reducedFeatVec.extend(featVec[axis+1:]) #将符合条件的添加到返回数据集

retDataSet.append(reducedFeatVec)

return retDataSet #返回划分后的数据集

chooseBestFeatureToSplit是选择最优特征的函数，函数返回的是信息增益最大的(最优)特征的索引值。你的任务是根据前面的公式，将有关计算信息增益部分的代码补充完整，首先需要获取划分后的子集，然后计算子集的概率，之后根据公式计算经验条件熵，再计算信息增益并打印每个特征的信息增益，最后需要找到最大的信息增益，并将其索引值返回。如果代码正确，可以得到如下结果值。对比计算的结果，会发现结果完全正确。最优特征的索引值为2，也就是特征A3(有自己的房子)。

def chooseBestFeatureToSplit(dataSet):

numFeatures = len(dataSet[0]) - 1 #特征数量

baseEntropy = calcShannonEnt(dataSet) #计算数据集的香农熵

bestInfoGain = 0.0 #信息增益

bestFeature = -1 #最优特征的索引值

vb=[]

for i in range(numFeatures):

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

uniqueVals = set(featList) #经验条件熵

# 计算信息增益

aum=0

for j in uniqueVals:

x=splitDataSet(dataSet,i,j)

l=len(x)

aum+=calcShannonEnt(x)\*(l/len(dataSet))

vb.append(baseEntropy-aum)

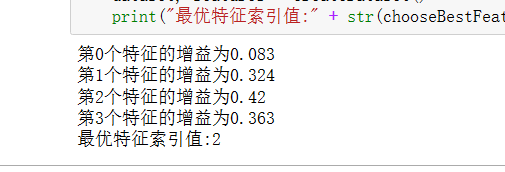
bestInfoGain=vb.index(max(vb))

return bestInfoGain

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

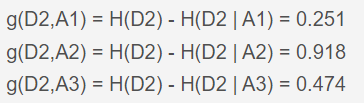
dataSet, features = createDataSet()

print("最优特征索引值:" + str(chooseBestFeatureToSplit(dataSet)))

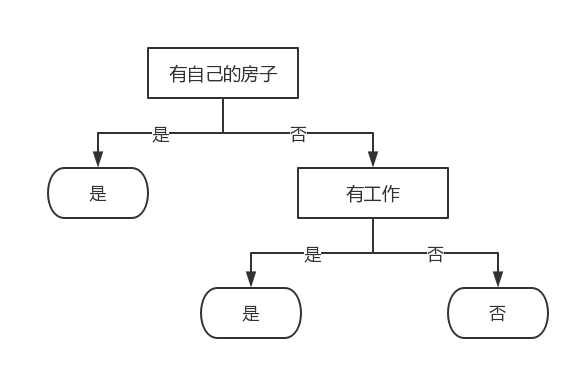


1. **构建决策树**

ID3算法的核心是在决策树各个结点上对应信息增益准则选择特征，递归地构建决策树。由于特征A3(有自己的房子)的信息增益值最大，所以选择特征A3作为根结点的特征。它将训练集D划分为两个子集D1(A3取值为"是")和D2(A3取值为"否")。由于D1只有同一类的样本点，所以它成为一个叶结点，结点的类标记为“是”。对D2则需要从特征A1(年龄)，A2(有工作)和A4(信贷情况)中选择新的特征，计算各个特征的信息增益：



根据计算，选择信息增益最大的特征A2(有工作)作为结点的特征。以此类推，这样就生成了一个决策树，该决策树只用了两个特征(有两个内部结点)，生成的决策树如下图所示。



我们使用字典存储决策树的结构，比如上小节我们分析出来的决策树，用字典可以表示为：{'有自己的房子': {0: {'有工作': {0: 'no', 1: 'yes'}}, 1: 'yes'}}。创建函数majorityCnt统计classList中出现此处最多的元素(类标签)。创建函数createTree用来递归构建决策树，其中参数featLabels表示存储选择的最优特征标签，返回的是构建的决策树。

递归创建决策树时，递归有两个终止条件：第一个停止条件是所有的类标签完全相同，则直接返回该类标签；第二个停止条件是使用完了所有特征，仍然不能将数据划分仅包含唯一类别的分组，即决策树构建失败，特征不够用。你需要将下面代码根据注释补充完整，最终可以输出我们构建的决策树：

def createTree(dataSet, labels, featLabels):

classList = set([example[-1] for example in dataSet])

if len(labels) == 0 or len(classList) == 1:

return majorityClass(dataSet)

bestFeat = chooseBestFeatureToSplit(dataSet)

bestFeatLabel = labels[bestFeat]

del(labels[bestFeat])

featLabels.append(bestFeatLabel)

featValues = set([example[bestFeat] for example in dataSet])

disct = {}

for value in featValues:

subLabels = labels.copy()

disct[value] = createTree(splitDataSet(dataSet, bestFeat, value), subLabels, featLabels)

return {bestFeatLabel: disct}

def majorityClass(dataSet):

classCount = {}

for example in dataSet:

classCount[example[-1]] = classCount.get(example[-1], 0) + 1

return max(classCount, key=classCount.get)

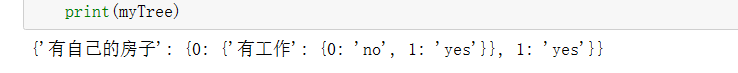
if \_\_name\_\_ == '\_\_main\_\_':

data = {}

dataSet, labels = createDataSet()

myTree = createTree(dataSet, labels, [])

print(myTree)



1. **使用决策树分类**

依靠训练数据构造了决策树之后，我们可以将它用于实际数据的分类。在执行数据分类时，需要决策树以及用于构造树的标签向量。然后，程序比较测试数据与决策树上的数值，递归执行该过程直到进入叶子结点；最后将测试数据定义为叶子结点所属的类型。在构建决策树的代码，可以看到，有个featLabels参数，它就是用来记录各个分类结点的，在用决策树做预测的时候，我们按顺序输入需要的分类结点的属性值即可。举个例子，比如用上节已经训练好的决策树做分类，那么只需要提供这个人是否有房子，是否有工作这两个信息即可，无需提供冗余的信息。

下面编写一个使用决策树分类的代码，输入测试数据[0,1]，它代表没有房子，但是有工作，测试分类结果为“放贷”。

def classify(inputTree, featLabels, testVec):

firstStr = next(iter(inputTree)) #获取决策树节点

secondDict = inputTree[firstStr] #下一个字典

featIndex = featLabels.index(firstStr)

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

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

dataSet, labels = createDataSet()

featLabels = []

myTree = createTree(dataSet, labels, featLabels)

testVec = [0,1] #测试数据

result = classify(myTree, featLabels, testVec)

if result == 'yes':

print('放贷')

if result == 'no':

print('不放贷')



1. **使用sklearn预测隐形眼镜类型**

一旦理解了决策树的工作原理，我们就可以帮助人们判断需要佩戴的镜片类型。隐形眼镜数据集是非常著名的数据集，它包含很多患者眼部状态的观察条件以及医生推荐的隐形眼镜类型。隐形眼镜类型包括硬材质(hard)、软材质(soft)以及不适合佩戴隐形眼镜(no lenses)。

数据集一共有24组数据，数据的Labels依次是age、prescript、astigmatic、tearRate、class，也就是第一列是年龄，第二列是症状，第三列是是否散光，第四列是眼泪数量，第五列是最终的分类标签。

sklearn.tree模块提供了决策树模型，用于解决分类问题和回归问题。我们使用DecisionTreeClassifie构建决策树，这个函数共有12个参数，详细使用说明可以参考sklearn官方使用手册。

下面我们编写代码：

# -\*- coding: UTF-8 -\*-

from sklearn import tree

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

fr = open('decision\_tree\_glass/lenses.txt')

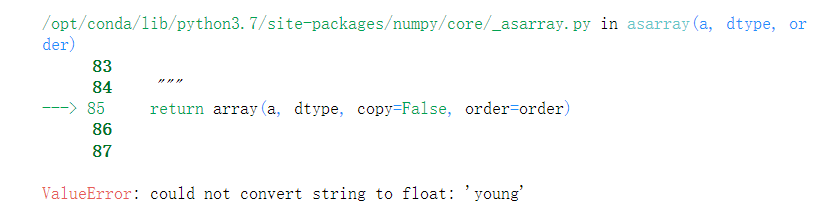
lenses = [inst.strip().split('\t') for inst in fr.readlines()]

print(lenses)

lensesLabels = ['age', 'prescript', 'astigmatic', 'tearRate']

clf = tree.DecisionTreeClassifier()

lenses = clf.fit(lenses, lensesLabels)



我们可以看到程序报错了，这是为什么？因为fit()函数不能接收string类型的数据，通过打印的信息可以看到，数据都是string类型的。在使用fit()函数之前，我们需要对数据集进行编码，这里可以使用两种方法：

　　LabelEncoder ：将字符串转换为增量值

　　OneHotEncoder：使用One-of-K算法将字符串转换为整数

　　为了对string类型的数据序列化，需要先生成pandas数据，这样方便我们的序列化工作。这里使用的方法是：原始数据->字典->pandas数据，下面将编写代码并进行序列化，从打印结果可以看到序列化后效果。

# -\*- coding: UTF-8 -\*-

from sklearn.preprocessing import LabelEncoder, OneHotEncoder

from six import StringIO

from sklearn import tree

import pandas as pd

import numpy as np

import pydotplus

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

with open('decision\_tree\_glass/lenses.txt', 'r') as fr:

lenses = [inst.strip().split('\t') for inst in fr.readlines()]

lenses\_target = [] #提取每组数据的类别，保存在列表里

for each in lenses:

lenses\_target.append(each[-1])

lensesLabels = ['age', 'prescript', 'astigmatic', 'tearRate'] #特征标签

lenses\_list = [] #保存lenses数据的临时列表

lenses\_dict = {} #保存lenses数据的字典，用于生成pandas

for each\_label in lensesLabels: #提取信息，生成字典

for each in lenses:

lenses\_list.append(each[lensesLabels.index(each\_label)])

lenses\_dict[each\_label] = lenses\_list

lenses\_list = []

#print(lenses\_dict)

lenses\_pd = pd.DataFrame(lenses\_dict)

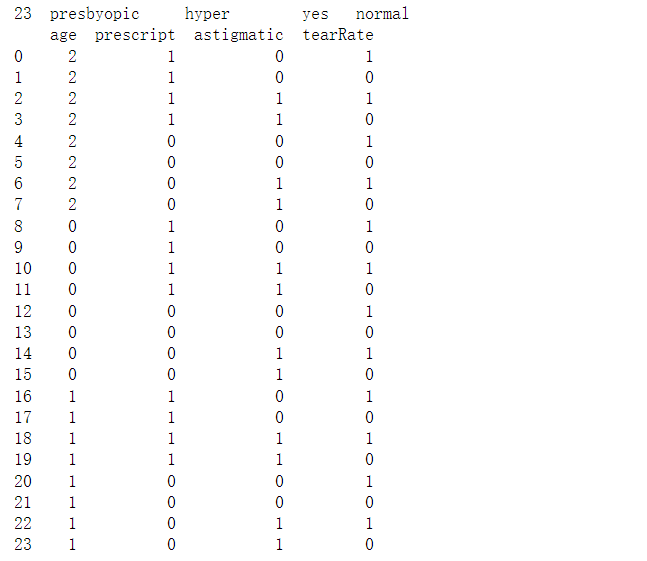
print(lenses\_pd)

le = LabelEncoder() #创建LabelEncoder()对象，用于序列化

for col in lenses\_pd.columns: #为每一列序列化

lenses\_pd[col] = le.fit\_transform(lenses\_pd[col])

print(lenses\_pd)



（图片省略部分输出）

我们接下来可以在上面代码基础上，继续进行决策树的构造和预测了。你可以根据自己的眼睛情况和年龄等特征，看一看自己适合何种材质的隐形眼镜。使用如下代码就可以看到预测结果：print(clf.predict([[1,1,1,0]]))。

详细代码如下，同时给出了绘制决策树并保存为pdf的相关代码，你可以敲写如下代码进行测试：

clf = tree.DecisionTreeClassifier(max\_depth = 4) #创建DecisionTreeClassifier()类

clf = clf.fit(lenses\_pd.values.tolist(), lenses\_target) #使用数据，构建决策树

dot\_data = StringIO()

tree.export\_graphviz(clf, out\_file = dot\_data, #绘制决策树

feature\_names = lenses\_pd.keys(),

class\_names = clf.classes\_,

filled=True, rounded=True,

special\_characters=True)

graph = pydotplus.graph\_from\_dot\_data(dot\_data.getvalue())

graph.write\_pdf("myTree.pdf")

print(clf.predict([[1,1,1,0]]))

