机器学习——决策树

个人觉得决策树这一部分的理论理解完全建立在信息论的基础上，因为我之前学过《信息论》，所以觉得很容易理解，如果没有信息论基础的同学其实挺建议去学一下，回过头来看你完全可以一下子就明白决策树的。

决策树是一种具有内部意义的分类树形结构，例如，对于某一个实例的测试，它用一颗树描述了每一个特征的if-then规则流程，就好比让你去爬树，首先会让你来判断，你左手有力还是右手有力，如果左手有力那你爬左边，如果右手有力你爬右边，做完了第一个选择你已经达到了第一层，接下来在你这个节点有新的选择，例如树枝左边断了还是右边断了，如果左边断了爬右边，右边断了爬左边，这样一步步达到叶节点，会给你一个答案——你爬树成功了没。他就是由一个个条件判断来组成的，我们的目标就是对这么多条件判断进行一个优先级的排序以及次序、关系的树构建。

关于怎么去判断这个流程树的关系构建就涉及到信息论的香农熵以及互信息增益，因为我本科是通信，所以对于信道容量用最大互信息来表示还是有很直接的理解的，不过要细讲清楚其实还是很多数学推导以及模型建立，如果感兴趣的或者想深入研究的可以参考《无线通信基础》里面的球体填充模型以及里面appendix.B的内容，讲得很好。

言归正传，回到互信息，我们怎么判断这些特征在构建树时的关系以及顺序呢？那就要看互信息了，这个也叫信息增益，信息增益越大，这个特征对于最终的分类结果影响越大，于是我们就可以从已有的特征中每次选出现有特征中互信息最大的一个作为节点，然后依次寻找接下来的节点就可以构造出最终的决策树了。

我们对一个实例进行实战演练，对于借贷进行一个决策，特征包括（有没有房子、有没有工作、信贷情况、年龄），我们构造这样的特征矩阵进行简单地构建：

```
from math import log
import operator

def calcShannonEnt(dataSet):
    numEntire=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])/numEntire
        shannonEnt-=prob*log(prob,2)
    return shannonEnt

def createDataSet():
    dataSet=[[0, 0, 0, 0, 'no'],
             [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']
    return dataSet,labels

def splitDataSet(dataSet,axis,value):
    returnDateSet=[]
    for featVec in dataSet:
        if featVec[axis]==value:
            reducedFeatVec=featVec[:axis]
            reducedFeatVec.extend(featVec[axis+1:])
            returnDateSet.append(reducedFeatVec)
    return returnDateSet

def chooseBest(dataSet):
    numFeatures=len(dataSet[0])-1
    baseEntropy=calcShannonEnt(dataSet)
    bestInforGain =0.0
    bestFeature=-1
    for i in range(numFeatures):
        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>bestInforGain):
            bestInforGain=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

sortClassCount=sorted(classCount.items(),key=operator.itemgetter(1),reverse=True
)
    return sortClassCount[0][0]

def createTree(dataSet,labels,featLabels):
    classlist=[example[-1] for example in dataSet]
    if classlist.count(classlist[0])==len(classlist):
        return classlist[0]
    if len(dataSet[0])==1 or len(labels)==0:
        return majorityCnt(classlist)
    bestFeat=chooseBest(dataSet)
    bestFeatLabel=labels[bestFeat]
    featLabels.append(bestFeatLabel)
    mytree={bestFeatLabel:{}}
```

```
    del(labels[bestFeat])
    featValues=[example[bestFeat] for example in dataSet]
    uniqueVals=set(featValues)
    for value in uniqueVals:
        mytree[bestFeatLabel]
[value]=createTree(splitDataSet(dataSet,bestFeat,value),labels,featLabels)
    return mytree

def classify(inputTree,featLabels,testVec):
    firstStr=next(iter(inputTree))
    secondDict=inputTree[firstStr]
    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

dataSet,labels=createDataSet()
featLabels=[]
mytree=createTree(dataSet,labels,featLabels)
print(mytree)
testVec=[0,1]
result=classify(mytree,featLabels,testVec)
if(result=='yes'):
    print('放贷')
else:
    print('不放贷')
```

> {'年龄': {0: {'有工作': {0: 'no', 1: 'yes'}}, 1: {'有自己的房子': {0: {'信贷情况': {0: 'no', 1: 'yes'}}, 1: 'yes'}}, 2: 'yes'}}
> 放贷

总的来说，使用白盒理论来解决决策树问题是很容易被理解的（如果对于互信息有一个比较好的理解），但是决策树也是比较不稳定的，比较依赖分类数据训练。