决策树实验报告

1711290 李涵 信息安全

一、问题描述

现有一组数据是对西瓜一些属性和西瓜好坏的分析结果，采用决策树的方法，根据这些属性，对西瓜的好坏进行预测。

二、解决方法

1.解决思路

数据集Watermelon-train1当中，西瓜的属性都是离散的，所以构造 ID3 决策树；数据集Watermelon-train2 中有连续型属性，所以构造C45决策树或CART决策树。

2.基本理论

决策树：

决策树是一个属性结构的预测模型，代表对象属性和对象值之间的一种映射关系，由节点和有向边组成，其节点有内节点和叶节点两种类型，内部节点表示一个特征或属性，叶节点表示一个类。 其主要优点是模型具有可读性，分类速度快，易于理解。

信息熵H(X)：

信息熵是香农在信息论中提出来的。熵的定义如下：

$$H(X) = -\sum_{i=1}^{n} p_i log\ p_i$$

X表示该事件取的有限个值的离散随机变量，pi表示每个随机变量在整个事件中的概率。分类的最终目的就是使信息熵最小，即通过特征可以最大概率地确定事件。

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，定义为：

$$H(Y|X) = \sum_{i=1}^{n} p_i H(Y|X = x_i)$$

pi表示变量中xi 的概率，H(Y|X=xi)是X=xi时Y的熵

信息增益g(Y,X)：

表示已知特征X的信息而使得类别Y的信息不确定性减少的程度，定义为：

$$g(Y, X) = H(Y) - H(Y|X)$$

H(Y)为样本类别Y的熵，H(Y|X) 为条件熵。

信息增益比gR(Y,X)：

信息增益率类似于归一化处理，不同之处归一化所用的信息是“分裂信息值”。在此，我们用信息熵来定义每个特征的熵，则最终的信息增益为：

$$gR(Y,X)=\frac{H(Y)-H(Y|X)}{H(X)}$$

如果出现上信息增益中所说的某类特征有很多值得情况，则特征X的不确定度很大，信息熵H(X)很大，会使整个信息增益比变小。

基尼指数：

假设有K个类，样本点属于第K的概率为pk，则概率分布的基尼指数为：

$$Gini(p)=\sum_{k=1}^{K}p_k(1-p_k)=1-\sum_{k=1}^{K}p_k^2$$

基尼指数与熵类似，都表示样本的不确定度。在CART算法中特征选择就是用的基尼指数。

ID3、C45、CART决策树分别依据信息增益、信息增益率和基尼指数来构建的。

剪枝：

在生成树的过程中，每一个叶都是单独的一类，对训练集是完全拟合的，但对测试集的预测效果不好，过拟合导致泛化能力不强。因此，要减掉一些枝叶，使得模型泛化能力更强。根据剪枝所出现的时间点不同，分为预剪枝和后剪枝。预剪枝是在决策树的生成过程中进行的；后剪枝是在决策树生成之后进行的。

剪枝策略包括：悲观错误剪枝 PEP (Pessimistic Error Pruning)、代价-复杂度剪枝 CCP (Cost-Complexity Pruning)、基于错误剪枝 EBP (Error-Based Pruning)、最小错误剪枝 MEP (Minimum Error Pruning)

3.算法流程

ID3决策树：

（1）选取能够得到最大信息增益的特征为数据划分归类

（2）直到全部划分结束而不对树的规模进行任何控制

（3）等树生成之后，执行后剪枝

C45决策树：

（1）选取能够得到最大信息增益率的特征来划分数据

（2）直到全部划分结束而不对树的规模进行任何控制

（3）等树生成之后，执行后剪枝

CART决策树：

（1）选取能够得到最小基尼指数的特征来划分数据

（2）直到全部划分结束而不对树的规模进行任何控制

（3）等树生成之后，执行后剪枝

构造决策树的步骤：

（1） 构建根节点，将所有训练数据都放在根节点，选择一个最优特征，按着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

（2） 如果这些子集已经能够被基本正确分类，那么构建叶节点，并将这些子集分到所对应的叶节点去

（3） 如果还有子集不能够被正确的分类，那么就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，如果递归进行，直至所有训练数据子集被基本正确的分类，或者没有合适的特征为止。

三、实验分析

1.实验数据

Watermelon-train1和Watermelon-test1有色泽、根蒂、敲声、纹理这四个离散属性和好瓜这一标签；

Watermelon-train2和Watermelon-test2有色泽、根蒂、敲声、纹理这四个离散属性、密度这一连续属性和好瓜这一标签。

注意：在处理数据的时候，要把编号这一列去除，防止决策树将编号作为分类标准

2.实验设计

ID3决策树：

计算香农熵

```
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*log(prob,2)
    return shannonEnt
```

根据某个特征的某个值对数据集进行划分

```
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
```

选择最好的数据集划分方式

```
def chooseBestFeatureToSplit(dataSet,labels):
    numFeatures=len(dataSet[0])-1
    baseEntropy=calcShannonEnt(dataSet)
    bestInfoGain=0.0
    bestFeature=-1
    bestSplitDict={}
    for i in range(numFeatures):
        featList=[example[i] for example in dataSet]

        uniqueVals=set(featList)
        newEntropy=0.0
        #计算该特征下每种划分的信息熵
        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
```

特征若已经划分完，节点下的样本还没有统一取值，则需要进行投票

```
def majorityCnt(classList):
    classCount={}
    for vote in classList:
        if vote not in classCount.keys():
            classCount[vote]=0
        classCount[vote]+=1
    return max(classCount)
```

生成决策树

```
def createTree(dataSet,labels,data_full,labels_full):
    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,labels)
    bestFeatLabel=labels[bestFeat]
    myTree={bestFeatLabel:{}}
    featValues=[example[bestFeat] for example in dataSet]
    uniqueVals=set(featValues)

    if type(dataSet[0][bestFeat]).__name__=='str':
```

```
        currentlabel=labels_full.index(labels[bestFeat])
        featValuesFull=[example[currentlabel] for example in data_full]
        uniqueValsFull=set(featValuesFull)

    del(labels[bestFeat])

    #针对bestFeat的每个取值，划分出一个子树。
    for value in uniqueVals:
        subLabels=labels[:]
        if type(dataSet[0][bestFeat]).__name__=='str':
            uniqueValsFull.remove(value)
        myTree[bestFeatLabel][value]=createTree(splitDataSet\
         (dataSet,bestFeat,value),subLabels,data_full,labels_full)

    if type(dataSet[0][bestFeat]).__name__=='str':
        for value in uniqueValsFull:
            myTree[bestFeatLabel][value]=majorityCnt(classList)

    return myTree
```

主程序

```
df=pd.read_csv('Watermelon-train1.csv',encoding='ANSI')
data=df.values[:,1:].tolist()
data_full=data[:]
labels=df.columns.values[1:-1].tolist()
labels_full=labels[:]
myTree=createTree(data,labels,data_full,labels_full)

df1=pd.read_csv('Watermelon-test1.csv',encoding='ANSI')
data1=df1.values[:,1:5].tolist()
reality=df1.values[:,5].tolist()
length=len(data1)
correct=0
i=0
for each in data1:
    tree=copy.copy(myTree)
    deep=0
    while 1:
        curr_feature=list(tree.keys())[0]
        all_feature_labels=list(list(tree.values())[0].keys())
        temp=list(list(tree.values())[0].values())
        p=0

        while 1:
            if all_feature_labels[p] in each:
                break
            p=p+1

        tree=temp[p]

        if '是' == tree or '否' ==tree:
```

```
            print(tree)
            if tree==reality[i]:
                correct=correct+1

            i=i+1
            break

print(correct/len(data1))
```

C45决策树：

对于连续的取值采用分裂点进行分裂

```
def splitContinuousDataSet(dataSet,axis,value,direction):
    retDataSet=[]
    for featVec in dataSet:
        if direction==0:
            if featVec[axis]>value:
                reducedFeatVec=featVec[:axis]
                reducedFeatVec.extend(featVec[axis+1:])
                retDataSet.append(reducedFeatVec)
        else:
            if featVec[axis]<=value:
                reducedFeatVec=featVec[:axis]
                reducedFeatVec.extend(featVec[axis+1:])
                retDataSet.append(reducedFeatVec)
    return retDataSet
```

在选择划分数据集的最好特征的时候，对于离散属性和连续属性分别进行处理，并且用增益率来代替增益，找到最好特征

```
def chooseBestFeatureToSplit(dataSet,labels):
    numFeatures=len(dataSet[0])-1
    baseEntropy=calcShannonEnt(dataSet)
    bestInfoGain=0.0
    bestFeature=-1
    bestSplitDict={}
    for i in range(numFeatures):
        featList=[example[i] for example in dataSet]
        #对连续型特征进行处理
        if type(featList[0]).__name__=='float' or type(featList[0]).__name__=='int':
            sortfeatList=sorted(featList)
            splitList=[]
            for j in range(len(sortfeatList)-1):
                splitList.append((sortfeatList[j]+sortfeatList[j+1])/2.0)

            bestSplitEntropy=10000
            slen=len(splitList)

            for j in range(slen):
                value=splitList[j]
```

```
                newEntropy=0.0
                subDataSet0=splitContinuousDataSet(dataSet,i,value,0)
                subDataSet1=splitContinuousDataSet(dataSet,i,value,1)
                prob0=len(subDataSet0)/float(len(dataSet))
                newEntropy+=prob0*calcShannonEnt(subDataSet0)
                prob1=len(subDataSet1)/float(len(dataSet))
                newEntropy+=prob1*calcShannonEnt(subDataSet1)
                if newEntropy<bestSplitEntropy:
                    bestSplitEntropy=newEntropy
                    bestSplit=j

            bestSplitDict[labels[i]]=splitList[bestSplit]
            infoGain=(baseEntropy-bestSplitEntropy)/baseEntropy

        #对离散型特征进行处理

        else:
            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)/baseEntropy
        if infoGain>bestInfoGain:
            bestInfoGain=infoGain
            bestFeature=i

    if type(dataSet[0][bestFeature]).__name__=='float' or type(dataSet[0]
[bestFeature]).__name__=='int':
        bestSplitValue=bestSplitDict[labels[bestFeature]]
        labels[bestFeature]=labels[bestFeature]+'<='+str(bestSplitValue)
        for i in range(shape(dataSet)[0]):
            if dataSet[i][bestFeature]<=bestSplitValue:
                dataSet[i][bestFeature]=1
            else:
                dataSet[i][bestFeature]=0

    return bestFeature
```

其他代码均与ID3类似

部分代码参考了机器学习实战（图灵程序设计丛书）

3.实验结果

ID3决策树对Watermelon-test1分类的准确率为70%

C45决策树对Watermelon-test2分类的准确率为20%