Author: 陆星宇 Date: 2021/2/9

第5章 决策树

一、交流讨论

问题1：为什么以信息增益作为划分训练数据集的特征，会偏向于选择取值较多的特征?

在训练数据集的经验熵大的时候，信息增益值会偏大，反之信息增益会偏小，使用信息增益比可以对这一问题进行校正。

问题2：cart剪枝中，“以t为单节点树”得到的损失函数是什么含义?

将t的左右子树剪枝后，t为根结点的树的损失函数。

问题3：ID3和C4.5两种算法生成的决策树还需要后剪枝吗

是的，因为终止条件始终是相对于训练集的，需要用测试集剪枝来尽量减小过拟合程度。

问题4：除了剪枝，还有没有什么方法可以提高决策的准确率

训练前，清洗数据；增加数据集的数量；

二、内容概要

1. 决策树模型

决策树由节点和有向边组成，结点分为内部结点和叶结点，内部结点表示一个特征或属性，叶结点表示一个类。由根结点到叶节点的每一条路径构成一条规则，规则之间互斥且完备。决策树表示给定特征条件下类的条件概率分布。分类时将结点的实例分到条件概率大的那一类去。

2. 决策树学习

决策树学习算法包含特征选择、决策树的生成与决策树的剪枝过程。

(1) 特征选择

特征选择的准则是信息增益或信息增益比。

a. 信息增益

设离散型随机变量X的概率分布为:

$$P(X=x_i)=p_i,\ i=1,2,\dots,n$$

则X的**熵**定义为:

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

熵越大，随机变量的不确定性越大。

从定义可以验证

$$0\le H(X)\le log\ n$$

定义**条件熵**为X给定条件下Y的条件概率分布对X的数学期望:

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

另外，令$0log0=0$。

定义特征A对训练数据集D的**信息增益**为：

$$g(D,A)=H(D)-H(D|A)$$

信息增益大的特征具有更强的分类能力。

信息增益算法如下：

1. 计算数据集D的信息熵H(D):

$$H(D)=-\sum_{k=1}^{K}\frac{|C_k|}{|D|}log_2\frac{|C_k|}{|D|}$$

2. 计算特征A对数据集D的经验条件熵H(D|A):

$$H(D|A)=\sum_{i=1}^{n}\frac{|D_i|}{|D|}H(D_i)=-\sum_{i=1}^{n}\frac{|D_i|}{|D|}\sum_{k=1}^{K}\frac{|D_{ik}|}{|D_i|}log_2\frac{|D_{ik}|}{|D_i|}$$

其中D_i表示根据A特征划分的D的子集，D_{ik}表示根据类别划分的D_i的子集。

3. 计算信息增益

$$g(D,A)=H(D)-H(D|A)$$

b. 信息增益比

在训练数据集的经验熵大的时候，信息增益值会偏大，反之信息增益会偏小，使用信息增益比可以对这一问题进行校正。

定义信息增益比为：

$$g_R(D,A)=\frac{g(D,A)}{H(D)}$$

(2) 决策树的生成

a. ID3算法

输入：

训练数据集D，特征集A，阈值e;

输出：决策树T.

1. 若D中所有实例属于同一类，则T为单节点数，并将该类作为该节点的类标记，返回T；
2. 若特征集为空，则T为单结点树，并将D中实例数最大的类作为类标记，返回T；
3. 否则，计算A中各特征对D的信息增益，选择信息增益最大的特征A_g;
4. 若A_g的信息增益小于阈值e，则置T为单节点树，并将D中实例数最大的类作为该节点的类标记，返回T；
5. 否则对A_g的每一个可能值a_i，依$A_g=a_i$将D分割为若干非空子集D_i，将D_i中实例数最大的类作为标记，构建子结点，由结点及其子节点构成树T，返回T;
6. 对第i各子结点，以D_i为训练集，以$A-A_g$为特征集，递归地调用Step1到Step5,得到子树T_i，并返回T_i

b. C4.5算法

与ID3算法相似，但将信息增益换成了信息增益比

(3) 决策树的剪枝

定义损失函数：

$$C_\alpha(T)=\sum_{t=1}^{|T|}N_tH_t(T)+\alpha|T|$$

其中$|T|$是树T叶结点的数量。

树的剪枝算法：

输入：生成算法生成的整个树，参数α;

输出：修建后的子树T_α.

1. 计算每个结点的经验熵；
2. 递归地从树地叶结点向上回缩，若会所后整体树的损失函数减小了(这一步可以局部计算)，则进行剪枝，其父结点变为新的结点；
3. 返回Step2直到不能继续为止。

3. CART算法

CART由特征选择，树的生成及剪枝组成，既可以用于分类也可以用于回归，CART假设决策树是二叉树，内部节点特征的取值为“是”(左分支)和“否”(右分支)。

(1) CART生成

a. 回归树的生成

假设已经将输入空间划分为M个单元$R_1,R_2,\dots,R_M$,并且在每个单元R_m上有一个固定的输出值c_m，于是回归树模型可表示为：

$$f(x)=\sum_{m=1}^{M}c_mI(x\in R_m)$$

可以用平方误差$\sum_{x_i\in R_m}(y_i-f(x_i))^2$来表示回归树对于训练数据的预测误差，用平方误差最小化的准则求解每个单元上的最优输出值。易知，单元R_m上的c_m的最优值$\hat{c}_m$是R_m上的所有输入对应输出的均值，即

$$\hat{c}_m=ave(y_i|x_i\in R_m)$$

最小二乘回归树生成算法：

输入：训练数据集D;

输出：回归树$f(x)$.

1. 选择最优切分变量j与切分点s，求解：

$$min[min\sum_{x_i\in R_1(j,s)}(y_i-c_1)^2+min\sum_{x_i\in R_2(j,s)}(y_i-c_2)^2]$$

遍历变量j，固定j扫描切分点s，选择使上式最小的j,s.

2. 用选定的对(j,s)划分区域并决定相应的输出值；
3. 继续对两个子区域调用Step1,Step2，直到满足停止条件；

4. 将输入空间划分为M个区域，生成决策树f(x)；

b. 分类树的生成

基尼指数：

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

$$Gini(D,A)=\frac{|D_1|}{|D|}Gini(D_1)+\frac{|D_2|}{|D|}Gini(D_2)$$

基尼系数表示不确定性，与熵相似。

CART生成算法：

输入：训练数据集D, 停止计算的条件；

输出：CART决策树.

从根结点开始，递归地对每个结点进行以下操作：

1. 计算Gini(p)，对每一个特征的每个可能取值，以是否来分割，计算Gini(D,A)；
2. 选择基尼系数最小的特征及其对应的的切分点作为最优特征与最优切分点，从现结点生成两个子结点；
3. 对两个子节点递归地调用Step1与Step2，直至满足停止条件；
4. 生成CART决策树.

(2) CART剪枝

CART剪枝算法：

输入：CART算法生成的决策树T_0;

输出：最优决策树T_α.

1. 设$k=0$，$T=T_0$;
2. 设$\alpha=+\infty$;
3. 自下而上地对各内部结点t计算$C(T_t)$，$|T_t|$以及

$$g(t)=\frac{C(t)-C(T_t)}{|T_t|-1}$$
$$\alpha=min(\alpha,g(t))$$

4. 自上而下地访问内部节点t，如果有$g(t)=\alpha$，进行剪枝，并对叶结点t以多数表决法决定其类，得到树T;
5. 设$k=k+1$，$\alpha_k=\alpha$，$T_k=T$;
6. 如果T不是由根结点单独构成地树，则回到Step4;
7. 采用交叉验证法在子树序列$T_0,T_1,\ldots,T_n$中选取最优子树T_α.

三、算法实现

```
import random as rd
# 按一定规则随机生成训练集
A = [[0,1],[0,1,2,3,4,5],[0,1,2,3,4,5,6,7,8,9,10]]
C = [0,1]
D = []
for i in range(1000):
    D.append([[rd.randint(0,1),rd.randint(0,5),rd.randint(0,10)]])
```

```
    if(rd.randint(0,9) == 0):
        if(D[i][0][1] + D[i][0][2] > 7):
            D[i].append(0)
        else:
            D[i].append(1)
    else:
        if(D[i][0][1] + D[i][0][2] > 7):
            D[i].append(1)
        else:
            D[i].append(0)
# 基尼指数
def Gini0(inC,inD):
    cn = len(inC)
    dn = len(inD)
    cin = [0 for i in range(cn)]
    for s in D:
        cin[s[1]] += 1
    sumc = 0
    for i in range(cn):
        sumc += cin[i] * cin[i]
    sumc /= dn * dn
    return 1 - sumc
def Gini1(ina,inav,inC,inD):
    D1 = []
    D2 = []
    for s in D:
        if(s[0][ina] == inav):
            D1.append(s)
        else:
            D2.append(s)
    result = len(D1) * Gini0(inC,D1) + len(D2) * Gini0(inC,D2)
    result /= len(D)
    return result
# 二叉树
class treenode:
    def __init__(self,inA,inC,inD,inroot):
        self.root = inroot
        self.gini = Gini0(inC,inD)
        self.divia = 0
        self.diviav = 0

        cn = len(inC)
        dn = len(inD)
        cin = [0 for i in range(cn)]
        for s in D:
            cin[s[1]] += 1
        self.type = 0
        for i in range(cn):
            if(cin[i]>cin[self.type]):
                self.type = i
        if(cin[self.type]/dn < 0.7):
            for i in range(len(inA)):
                for j in inA[i]:
                    if(Gini1(i,j,inC,inD)
<Gini1(self.divia,self.diviav,inC,inD)):
                        self.divia = i
                        self.diviav = j
            D1 = []
```

```
            D2 = []
            for s in D:
                if(s[0][self.divia] == self.diviav):
                    D1.append(s)
                else:
                    D2.append(s)
            if(len(D1) == 0 or len(D2) == 0):
                self.left = None
                self.right = None
                self.isleaf = 1
            else:
                self.left = treenode(inA,inC,D1,self)
                self.right = treenode(inA,inC,D2,self)
                self.isleaf = 0
        else:
            self.isleaf = 1
    def Classify(self,ins):
        if(self.isleaf == 1):
            return self.type
        else:
            if(ins[0][self.divia] == self.diviav):
                return self.left.Classify(ins)
            else:
                return self.right.Classify(ins)
# 训练与测试
DTrain = D[0:699]
DTest = D[700:999]
rightnum = 0
CART = treenode(A,C,DTrain,None)
for item in DTest:
    if(CART.Classify(item) == item[1]):
        rightnum += 1
print("测试正确率为：",rightnum/len(DTest)*100,"%")
```

四、实验结果

```
测试正确率为：  54.84949832775919 %
```

分析：结果相当差，和瞎猜差不多。有两方面的原因：首先由于个人算法空间复杂度很高，所以为了减少二叉树深度，终止条件较宽；其次是没有剪枝操作，过拟合严重。

五、阅读拓展

无

六、阅读计划

《统计学习方法》第五章