机器学习——CART算法的学习

之前我们学会了决策树的原理以及使用ID3算法实现决策树的构建，我们知道决策树其实就是一种贪心算法，总是在当前情况作出最佳选择，并没有考虑到全局的最优选择，同时ID3算法对特征的选择总是分顺序的，一旦被选择后，在之后的执行就不会起任作用，这样的切分特征方式被认为是过于迅速，同时它对于连续性的数据特征是无法处理的，因此我们有了CART算法。

CART算法：

CART算法适用于连续性特征，它使用二分切元的方法来处理连续性变量，也就是判断特征值大于阈值就走左子树，否则就走右子树。因此更仔细地分析就是：

（1）决策树的生成：递归构建二叉决策树，自上而下从根构建节点，每个节点要选择一个最好的属性来分裂。

（2）决策树的剪枝：用验证数据集对已生成的树进行剪枝并选择最优子树，剪枝的标准就是损失函数最小。

其实我们之前的ID3使用信息熵来计算最佳切分特征，CART算法也是通过选择特征然后去切分数据集合进而构建树。我们使用训练集：

$$D=\{(x_1,y_1),(x_2,y_2),(x_3,y_3),\ldots,(x_n,y_n)\}$$

得到的回归树模型：

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

对于这个模型我们需要最小化因此有：

$$min\ error=\sum_{x_i\in R_m}(y_i-f(x_i))^2$$
$$\hat{c_m}=ave(y_i|x_i\in R_m)$$

接下来就是切分单元，如果我们选择x_j为切分变量，它的取值s是切分点，会得到两个区域：

$$R_1(j,s)=\{x|x^{(j)}\leq s\},R_2(j,s)=\{x|x^{(j)}>s\}$$

当j和s固定时，我们要找到两个区域的代表值c1，c2使得平方差最小：

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

同时我们已知c为区间平均，因此对于固定的j只需要找到最优的s，然后通过遍历所有变量，就可以找到最优的j，就可以得到最优对（j，s），并得到两个区间。这样的方法也叫最小二乘回归树。除此之外，我们需要定义两个参数，分别控制误差变化限制和切分特征最少样本数，这两个参数的意义是为了防止过拟合，提前设置终止条件。

树剪枝

一棵树如果它的节点过多，表明该模型可能对数据进行了“过拟合”，我们需要通过剪枝来避免过拟合，我们所说到的两个控制误差变化和切分特征的参数就是一种剪枝，但是这是预剪枝，这种阈值剪枝对于误差的数量级很敏感，我们需要不断寻找一个合适的值来确定什么样的结果才是最优的，因此对于一个多维度的数据集是很困难的。我们需要一种更好的后剪枝方法：

我们首先需要对数据集分成训练集和测试集，指定参数，使得构建的树足够的大、复杂，以便于我们剪枝，然后从上而下找到叶节点，用测试集判断这些节点是否能降低测试集误差，如果是就合并。我们的实战就是通过ex2.txt文件进行训练回归树，然后利用ex2test.txt对回归树进行剪枝，我们需要测试输入是否一颗树，返回一个布尔类型的结果，然后使用递归函数遍历计算平均值之后进行剪枝。

```
import numpy as np

def loadDataSet(filename):
    dataMat=[]
    fr = open(filename)
    for line in fr.readlines():
        curLine=line.strip().split('\t')
        fltLine=list(map(float,curLine))
        dataMat.append(fltLine)
    return dataMat

def binSplitDataSet(dataSet,feature,value):
    mat0=dataSet[np.nonzero(dataSet[:,feature]>value)[0],:]
    mat1=dataSet[np.nonzero(dataSet[:,feature]<=value)[0],:]
    return mat0,mat1

def regLeaf(dataSet):
    #生成叶节点(均值)
    return np.mean(dataSet[:,-1])

def regErr(dataSet):
    #误差估计函数(方差)
    return np.var(dataSet[:,-1])*np.shape(dataSet)[0]

def chooseBestSplit(dataSet,leafType=regLeaf,errType=regErr,ops=(1,4)):
    import types
    #tols允许的误差下降值，tolN切分的最少样本数
    tolS=ops[0]
    tolN=ops[1]
    #如果所有值相等，退出
    if len(set(dataSet[:,-1].T.tolist()[0]))==1:
        return None,leafType(dataSet)
    #默认最后一个特征为最佳切分，计算误差
    m,n=np.shape(dataSet)
    S=errType(dataSet)
    bestS=float('inf')
    bestIndex=0
    bestValue=0
    for featIndex in range(n-1):
        for splitVal in set(dataSet[:,featIndex].T.A.tolist()[0]):
            mat0,mat1=binSplitDataSet(dataSet,featIndex,splitVal)
            if(np.shape(mat0)[0]<tolN) or (np.shape(mat1)[0]<tolN):
                continue
            newS=errType(mat0)+errType(mat1)
            if newS<bestS:
                bestIndex=featIndex
                bestValue=splitVal
                bestS=newS
    if(S-bestS)<tolS:
        return None,leafType(dataSet)
    mat0,mat1=binSplitDataSet(dataSet,bestIndex,bestValue)
    if(np.shape(mat0)[0]<tolN) or (np.shape(mat1)[0]<tolN):
```

```
        return None,leafType(dataSet)
    return bestIndex,bestValue

def creatTree(dataSet,leafType=regLeaf,errType=regErr,ops=(1,4)):
    feat,val=chooseBestSplit(dataSet,leafType,errType,ops)
    if feat==None:
        return val
    retTree={}
    retTree['spInd']=feat
    retTree['spVal']=val
    lSet,rSet=binSplitDataSet(dataSet,feat,val)
    retTree['left']=creatTree(lSet,leafType,errType,ops)
    retTree['right']=creatTree(rSet,leafType,errType,ops)
    return retTree

def isTree(obj):
    import types
    return (type(obj).__name__=='dict')

def getMean(tree):
    if isTree(tree['right']):
        tree['right']=getMean(tree['right'])
        tree['left']=getMean(tree['left'])
    return (tree['left']+tree['right'])/2.0

def prune(tree,testData):
    if np.shape(testData)[0]==0:
        return getMean(tree)
    if (isTree(tree['right']) or isTree(tree['left'])):
        lSet,rSet=binSplitDataSet(testData,tree['spInd'],tree['spVal'])
    if isTree(tree['left']):
        tree['left']=prune(tree['left'],lSet)
    if isTree(tree['right']):
        tree['right']=prune(tree['right'],rSet)
    if not isTree(tree['left']) and not isTree(tree['right']):
        lSet,rSet=binSplitDataSet(testData,tree['spInd'],tree['spVal'])
        errorNomerge=np.sum(np.power(lSet[:,-1]-
tree['left'],2))+np.sum(np.power(rSet[:,-1]-tree['right'],2))
        treeMean=(tree['left']+tree['right'])/2.0
        errMerge=np.sum(np.power(testData[:,-1]-treeMean,2))
        if errMerge<errorNomerge:
            return treeMean
        else:
            return tree
    else:
        return tree


train_filename='ex2.txt'
train_data=loadDataSet(train_filename)
train_Mat=np.mat(train_data)
tree=creatTree(train_Mat)
print('剪枝前：')
print(tree)
test_filename='ex2test.txt'
test_Data=loadDataSet(test_filename)
test_Mat=np.mat(test_Data)
print('\n 剪枝后：')
```

```
print(prune(tree,test_Mat))
```

剪枝前：
{'spInd': 0, 'spVal': 0.499171, 'left': {'spInd': 0, 'spVal': 0.729397, 'left': {'spInd': 0, 'spVal': 0.952833, 'left': {'spInd': 0, 'spVal': 0.958512, 'left': 105.24862350000001, 'right': 112.42895575000001}, 'right': {'spInd': 0, 'spVal': 0.759504, 'left': {'spInd': 0, 'spVal': 0.790312, 'left': {'spInd': 0, 'spVal': 0.833026, 'left': {'spInd': 0, 'spVal': 0.944221, 'left': 87.3103875, 'right': {'spInd': 0, 'spVal': 0.85497, 'left': {'spInd': 0, 'spVal': 0.910975, 'left': 96.452867, 'right': {'spInd': 0, 'spVal': 0.892999, 'left': 104.825409, 'right': {'spInd': 0, 'spVal': 0.872883, 'left': 95.181793, 'right': 102.25234449999999}}}, 'right': 95.27584316666666}}, 'right': {'spInd': 0, 'spVal': 0.811602, 'left': 81.110152, 'right': 88.78449880000001}}, 'right': 102.35780185714285}, 'right': 78.08564325}}, 'right': {'spInd': 0, 'spVal': 0.640515, 'left': {'spInd': 0, 'spVal': 0.666452, 'left': {'spInd': 0, 'spVal': 0.706961, 'left': 114.554706, 'right': {'spInd': 0, 'spVal': 0.698472, 'left': 104.82495374999999, 'right': 108.92921799999999}}, 'right': 114.1516242857143}, 'right': {'spInd': 0, 'spVal': 0.613004, 'left': 93.67344971428572, 'right': {'spInd': 0, 'spVal': 0.582311, 'left': 123.2101316, 'right': {'spInd': 0, 'spVal': 0.553797, 'left': 97.20018024999999, 'right': {'spInd': 0, 'spVal': 0.51915, 'left': {'spInd': 0, 'spVal': 0.543843, 'left': 109.38961049999999, 'right': 110.979946}, 'right': 101.73699325000001}}}}}, 'right': {'spInd': 0, 'spVal': 0.457563, 'left': {'spInd': 0, 'spVal': 0.467383, 'left': 12.50675925, 'right': 3.4331330000000007}, 'right': {'spInd': 0, 'spVal': 0.126833, 'left': {'spInd': 0, 'spVal': 0.373501, 'left': {'spInd': 0, 'spVal': 0.437652, 'left': -12.558604833333334, 'right': {'spInd': 0, 'spVal': 0.412516, 'left': 14.38417875, 'right': {'spInd': 0, 'spVal': 0.385021, 'left': -0.8923554999999995, 'right': 3.6584772500000016}}}, 'right': {'spInd': 0, 'spVal': 0.335182, 'left': {'spInd': 0, 'spVal': 0.350725, 'left': -15.08511175, 'right': -22.693879600000002}, 'right': {'spInd': 0, 'spVal': 0.324274, 'left': 15.05929075, 'right': {'spInd': 0, 'spVal': 0.297107, 'left': -19.9941552, 'right': {'spInd': 0, 'spVal': 0.166765, 'left': {'spInd': 0, 'spVal': 0.202161, 'left': {'spInd': 0, 'spVal': 0.217214, 'left': {'spInd': 0, 'spVal': 0.228473, 'left': {'spInd': 0, 'spVal': 0.25807, 'left': 0.40377471428571476, 'right': -13.070501}, 'right': 6.770429}, 'right': -11.822278500000001}, 'right': 3.4496025}, 'right': {'spInd': 0, 'spVal': 0.156067, 'left': -12.1079725, 'right': -6.247900000000001}}}}}, 'right': {'spInd': 0, 'spVal': 0.084661, 'left': 6.509843285714284, 'right': {'spInd': 0, 'spVal': 0.044737, 'left': -2.544392714285715, 'right': 4.091626}}}}}

剪枝后：
{'spInd': 0, 'spVal': 0.499171, 'left': {'spInd': 0, 'spVal': 0.729397, 'left': {'spInd': 0, 'spVal': 0.952833, 'left': {'spInd': 0, 'spVal': 0.958512, 'left': 105.24862350000001, 'right': 112.42895575000001}, 'right': {'spInd': 0, 'spVal': 0.759504, 'left': {'spInd': 0, 'spVal': 0.790312, 'left': {'spInd': 0, 'spVal': 0.833026, 'left': {'spInd': 0, 'spVal': 0.944221, 'left': 87.3103875, 'right': {'spInd': 0, 'spVal': 0.85497, 'left': {'spInd': 0, 'spVal': 0.910975, 'left': 96.452867, 'right': {'spInd': 0, 'spVal': 0.892999, 'left': 104.825409, 'right': {'spInd': 0, 'spVal': 0.872883, 'left': 95.181793, 'right': 102.25234449999999}}}, 'right': 95.27584316666666}}, 'right': {'spInd': 0, 'spVal': 0.811602, 'left': 81.110152, 'right': 88.78449880000001}}, 'right': 102.35780185714285}, 'right': 78.08564325}}, 'right': {'spInd': 0, 'spVal': 0.640515, 'left': {'spInd': 0, 'spVal': 0.666452, 'left': {'spInd': 0, 'spVal': 0.706961, 'left': 114.554706, 'right': 106.87708587499999}, 'right': 114.1516242857143}, 'right': {'spInd': 0, 'spVal': 0.613004, 'left': 93.67344971428572, 'right': {'spInd': 0, 'spVal': 0.582311, 'left': 123.2101316, 'right': 101.580533}}}}, 'right': {'spInd': 0, 'spVal': 0.457563, 'left': 7.969946125, 'right': {'spInd': 0, 'spVal': 0.126833, 'left': {'spInd': 0, 'spVal': 0.373501, 'left': {'spInd': 0, 'spVal': 0.437652, 'left': -12.558604833333334, 'right': {'spInd': 0, 'spVal': 0.412516, 'left': 14.38417875, 'right': 1.383060875000001}}, 'right': {'spInd': 0, 'spVal': 0.335182, 'left': {'spInd': 0, 'spVal': 0.350725, 'left': -15.08511175, 'right': -22.693879600000002}, 'right': {'spInd': 0, 'spVal': 0.324274, 'left': 15.05929075, 'right': {'spInd': 0, 'spVal': 0.297107, 'left': -19.9941552, 'right': {'spInd': 0,

'spVal': 0.166765, 'left': {'spInd': 0, 'spVal': 0.202161, 'left': -5.801872785714286, 'right': 3.4496025}, 'right': {'spInd': 0, 'spVal': 0.156067, 'left': -12.1079725, 'right': -6.247900000000001}}}}}, 'right': {'spInd': 0, 'spVal': 0.084661, 'left': 6.509843285714284, 'right': {'spInd': 0, 'spVal': 0.044737, 'left': -2.544392714285715, 'right': 4.091626}}}}}
[Finished in 1.7s]

从这里看到确实剪枝有了一定的效果，但没有像预期那样剪枝成两部分，说明后剪枝可能都没有预剪枝好，一般采用两种技术同时结合。