2.3 递归建树

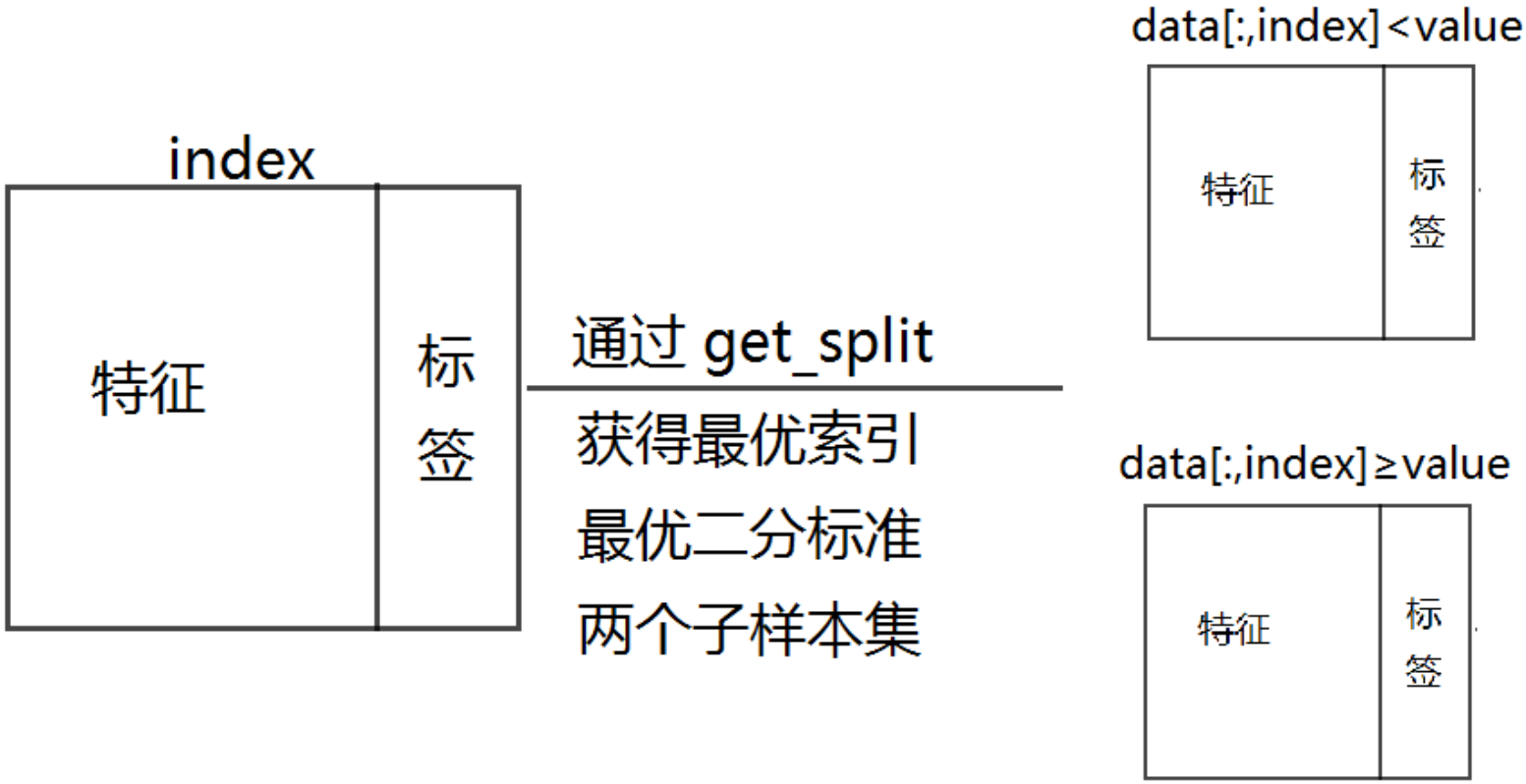
添加 create_tree(data,max_depth,min_size,depth) 函数
create_tree 考虑三个问题：
1. 终止条件
2. 递归过程
3. 节点结构

1. 终止条件：
与 ID3 不同，此时的特征不会减少
如果样本标签全部相同就停止
如果样本的数量少于给定阈值 **(3 或 5)** 就停止
如果树的深度大于给定阈值 **(8 或 10)** 就停止

停止的意思是从决策节点变成叶子节点并跳出
因此也需要一个 to_leaf_node 函数，
输入决策点的所有标签，返回出现次数最多的标签

2.3 递归过程

如何将总体样本集分割为两个小规模的问题：



2.3 节点结构

节点结构：使用什么形式保存一棵树，保存哪些信息

1. 使用字典来保存
2. 保存决策点的 特征索引，二分标准，左子树，右子树，左右子样本集

注意： get_split 函数返回的 groups 是左右样本集，并不是左右子树

1. tree={'index':index, 'value':value, 'left':{},'right':{} , 'groups':groups}
2. 如何生成子树：(create_tree(data))

   如果不满足终止条件:

   index , value , groups=get_split(data)

   tree={…}

   tree['left']=createTree(left)

   tree[‘right']=createTree(right)

2.4 建树步骤

create_tree 函数:
输入：样本集 data， 最大深度 max_depth， 最小分割样本数 min_size， 当前深度 depth
输出：决策树 tree
求得 data 的所有标签 label_list
如果标签个数少于 min_size 或者全部相同　或达到指定深度:
　　返回 label_list 中出现最频繁的标签
否则:
　　获得最优特征索引和最优二分标准 (调用 get_split 函数)
　　创建一棵树 (节点) tree
　　树的左分支递归调用 create_tree 函数，深度加 1
　　树的左分支递归调用 create_tree 函数，深度加 1
注意：树分支出的递归没有 return

2.4 建树步骤

终止条件添加了两种条件

```
create_tree (data,max_depth=999,min_size=1,depth=1):
    求得 data  的所有标签 labelList
    如果 labelList 的长度小于等于 min_size
    or labelList 中标签全一样
    or  此时的树深度 depth 到达了 max_depth :
        返回 labelList 中出现最多的标签
    index,value,groups=get_split(data)
    tree={'index':index,'value':value,'left':{},'right':{}}
     tree['left']=createTree(left,max_depth,minsize,depth+1)
     tree['right']=createTree(right,max_depth,minsize,depth+1)
```

小练习

2.4 小练习

CART_Classifier.py 里面应该有以下几个函数

gini ： 计算基尼系数和加权平均基尼系数

split_data ： 根据特征索引和二分标准对数据进行分割

get_split ： 得到数据集的最优分割特征索引和最优二分标准

create_tree： 利用递归生成树，并控制树深度

to_leaf_node ： 生成叶子节点 (还没给)

fit： 给定控制条件，调用递归，生成树 **(** 此处可以不写 **)**

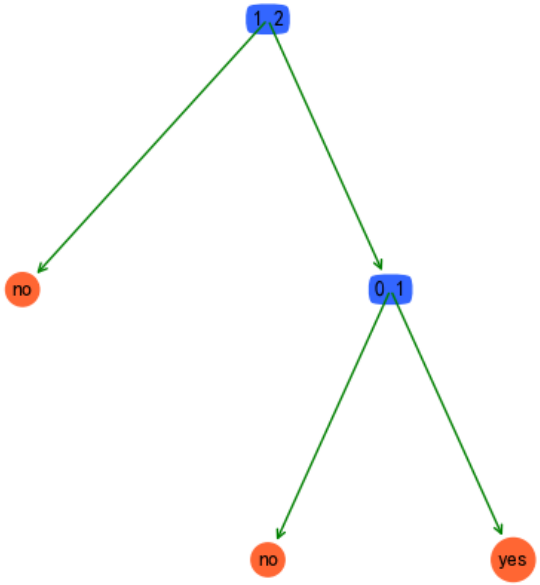
训练集为 dataSet

```
dataSet = [[1,2,'yes'],
           [1,2,'yes'],
           [1,0,'no'],
                   [1,1,'no'],
                   [0,2,'no']]
```

对训练集进行建树并打印树字典

3 CART 分类预测

{'index': 1, 'value': 2, 'left': 'no', 'right': {'index': 0, 'value': 1, 'left': 'no', 'right': 'yes'}}



predict # 根据树预测给定样本的标签
测试集 test
test = [[1,0],
[1,2]]
通过 CART 进行预测

predict(tree , sample)
和 ID3 的解树类似，但此时的子树就在 'left' 和 'right' 对应的值中，比 ID3 解树过程少一层

3.1 CART 分类预测

predict 函数：取一个样本 sample，利用 tree 对其预测
输入：树 tree，样本 sample
输出：预测类别
首先找到树最外层的 index 和 value
如果样本的第 index 维特征小于 value: #(考虑左子树)
    如果左子树是字典 (isinstance) :
        将 tree 更换为左子树；然后递归并返回
    否则:
        返回左子树的值　　# 即为预测值
否则:　　# 考虑右子树
    如果右子树是字典:
        将 tree 更换为右子树；然后递归并返回
    否则:
        返回右子树的值　　# 即为预测值

3.2 练习

所用数据集：水雷 - 岩石数据
文件名为： sonar.csv
该 csv 文件不包含 header(列名)，每一行代表一个个体；
文本共 61 个字段，前 60 个字段是特征字段为数值型连续变量，不存在缺失，不包括 ID 号列；
文件的最后一列为标签为字符型，表示样本的实际结果 :M 表示水雷， R 表示岩石。

```
[[0.02 0.0371 0.0428 ..., 0.009 0.0032 'R']
 [0.0453 0.0523 0.0843 ..., 0.0052 0.0044 'R']
 [0.0262 0.0582 0.1099 ..., 0.0095 0.0078 'R']
 ...,
 [0.0522 0.0437 0.018 ..., 0.0077 0.0031 'M']
 [0.0303 0.0353 0.049 ..., 0.0036 0.0048 'M']
 [0.026 0.0363 0.0136 ..., 0.0061 0.0115 'M']]
```

3.2 数据读入

常用的三种方法:

open ， pandas.read_csv ， pandas.read_excel

open:txt,csv,excel; read_csv:csv ; read_excel:excel

```
data=[]
with open('sonar.all-data.csv', 'r') as file:
    for row in file:
        if not row:
            continue
        data.append(row.strip().split(','))
```

但是此时返回的每个值都是字符型，需要再转换成 float, 或者 int

```
df = pd.read_csv(fileName , header=None,index_col=None)
data = df.values
# 会根据数据的类型自动转换， int , float , str
```

3.3 建树与预测

1. 首先将数据打乱： np.random.shuffle(data)
2. 将数据集分为训练和验证两部分 :

train=data[:180] ; val=data[180:]

train_d=[d[:-1] for d in train] ; train_l=[d[-1] for d in train]

val_d=[d[:-1] for d in val] ; val_l=[d[-1] for d in val]

3. 进行训练

tree=fit(train,max_depth=10,min_size=2)

4. 进行预测

pre=[predict(tree,d) for d in val_d]

5. 计算正确率

accuracy = (pre==val_l).tolist().count(true)/len(val_l)

4 CART 解决回归问题

如果标签列的值不是离散的，这时使用基尼系数或者熵就无法计算不纯度，因此需要新的公式计算连续标签的不纯度：最小平方误差和最小绝对误差

最小平方误差：

$$\sum_{x_i<p}\left(y_i-c_{jp}^{(1)}\right)^2+\sum_{x_i\geq p}\left(y_i-c_{jp}^{(2)}\right)^2$$

其中：$c_{jp}^{(1)}=\text{avg}(y_i \mid x_i<p)$、$c_{jp}^{(2)}=\text{avg}(y_i \mid x_i \geq p)$

最小绝对误差

$$\sum_{x_i<p}\left|y_i-c_{jp}^{(1)}\right|+\sum_{x_i\geq p}\left|y_i-c_{jp}^{(2)}\right|$$

4.1 平方误差实现：

根据 groups=[left,right] ，计算均方误差
代替了分类任务中的 gini 系数的计算

```
mean_square_error(groups):
m_s_e=0.0
for group in groups:
    size=len(group)
    if size==0:
        continue
    labels=[row[-1] for row in group]
    proportion=np.array(labels).mean()
    error=sum(power(labels - proportion,2))
    m_s_e+=error
    return m_s_e
```

4.2 停止条件和变成叶子节点：

分类预测时，其中一个停止条件是：如果全部的标签都是一样的那么返回

如果是连续标签，标签值全部相同几乎不可能

设定最小平方误差阈值 stop_mse 代替以上停止条件：

如果该分支下标签的最小均方误差 <stop_mse:

将该分支标签的均值作为叶子节点

决策节点转换为叶子节点

toLeafNode(classLabels)：

返回 classLabels 的均值

4.3 正确率改为均方误差：

如何判断模型的性能
分类预测：正确率，召回率
回归预测：均方误差（Mean Squar Error）

使用均方误差判定模型性能

$$\frac{1}{m}\sum_{i=1}^{m}(y_i-\overline{y}_i)^2$$

其中 y_i 是真实的标签数据，$\overline{y}_i$ 是预测的标签数据

4.4 回归树：

1. mean_squar_error(groups) 计算最小均方误差 # 修改
2. split(data,index,value) 划分数据集
3. get_split(data) 获得最优特征与二分标准
4. toLeafNode(labelList) 变成叶子节点 # 需要修改的
5. createTree(data,max_depth,min_size,depth,stop) 递归建树，每个节点记录 {index， value，left， right， stop} # 需要添加 stop
6. predict(tree ,example) 递归解树，进行预测
7. m_s_e(tree, test_data, test_label) 计算预测的均方误差

4.5 回归树预测

能不能利用回归树对水雷 - 岩石数据集进行分类?

将水雷作为数值 0，岩石作为数值 1 进行回归
然后预测值小于 0.5 是水雷，大于 0.5 是岩石

0.5 的设定合理吗？能不能选别的？选不同的值
会有什么结果?

ROC 曲线和 AUC 面积是怎么回事

4.1 水雷 - 岩石数据

将水雷作为数值 0，岩石作为数值 1 进行回归
然后预测值小于 0.5 是水雷，大于 0.5 是岩石