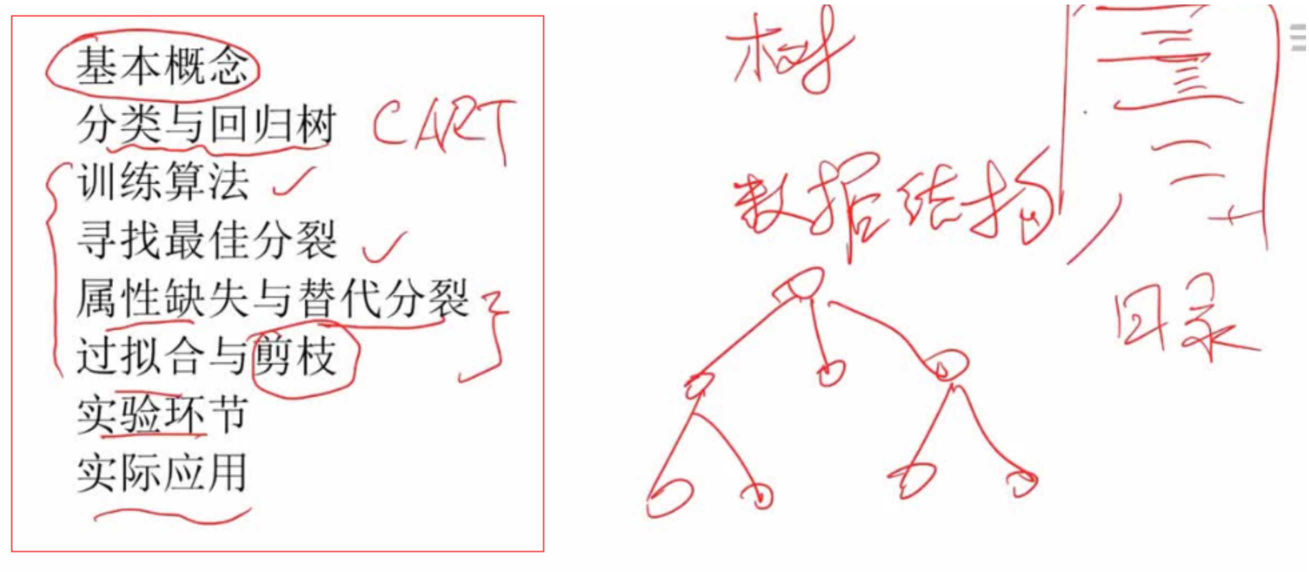
6 决策树

重点：训练算法的核心问题。

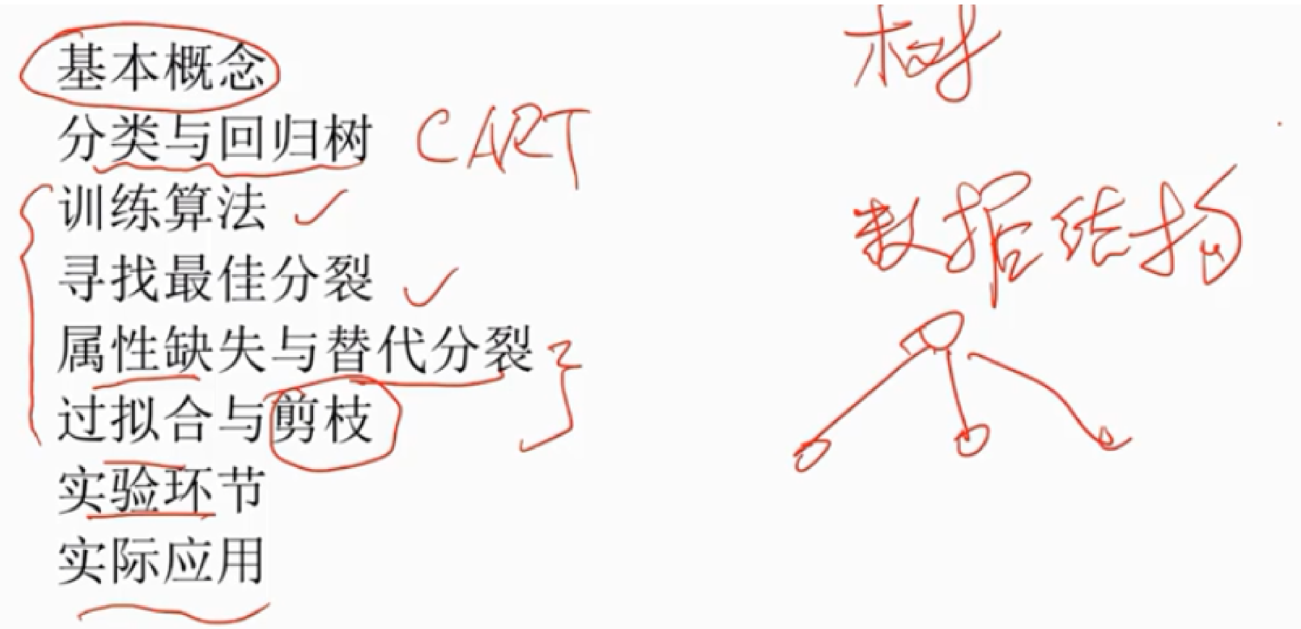
难点：递归分裂过程。

6.1 本集内容介绍



6.2 树与二叉树简介

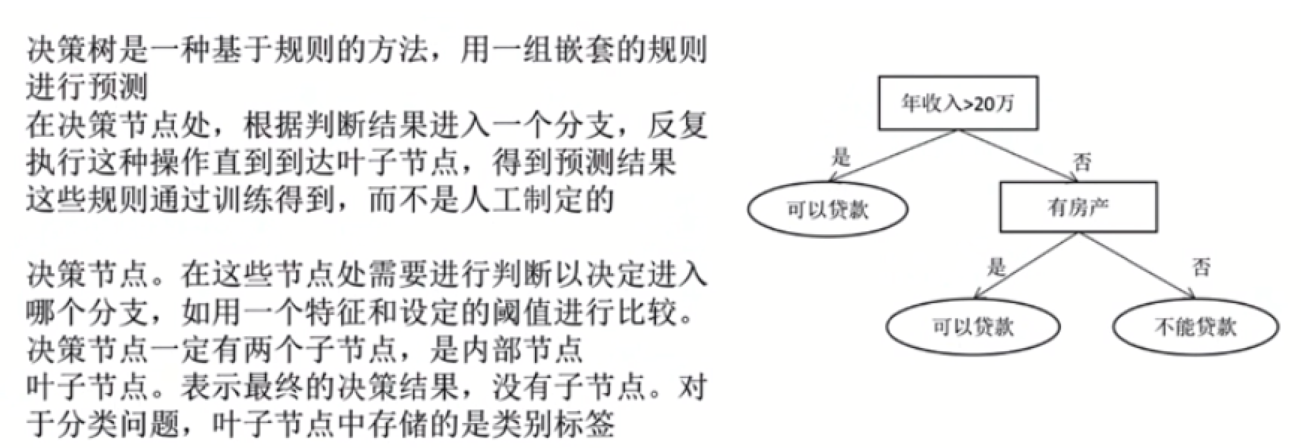
补充数据结构中的树。



树的例子：你的家族，书的目录。

树是一个分层的结构，还有递归的结构。

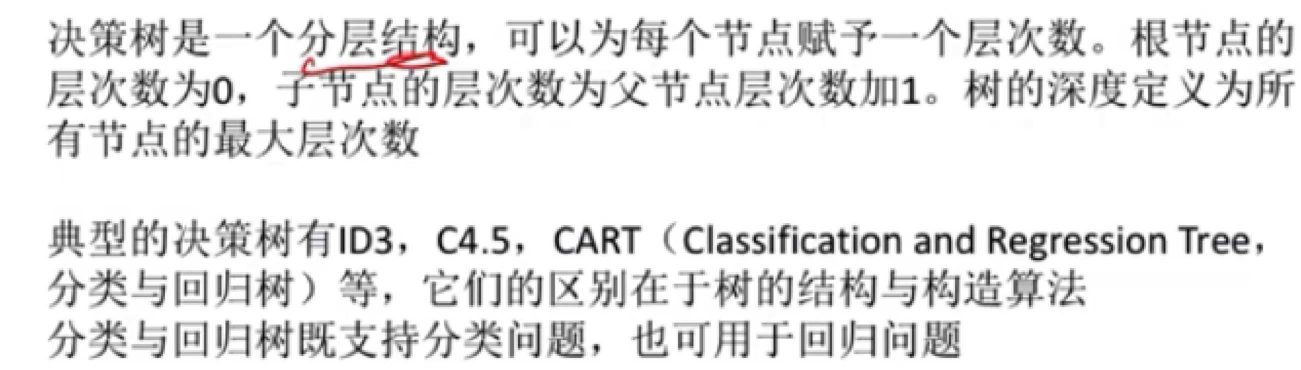
6.3 决策树简介



决策树是一种基于规则的方法，用一组嵌套的规则进行预测。

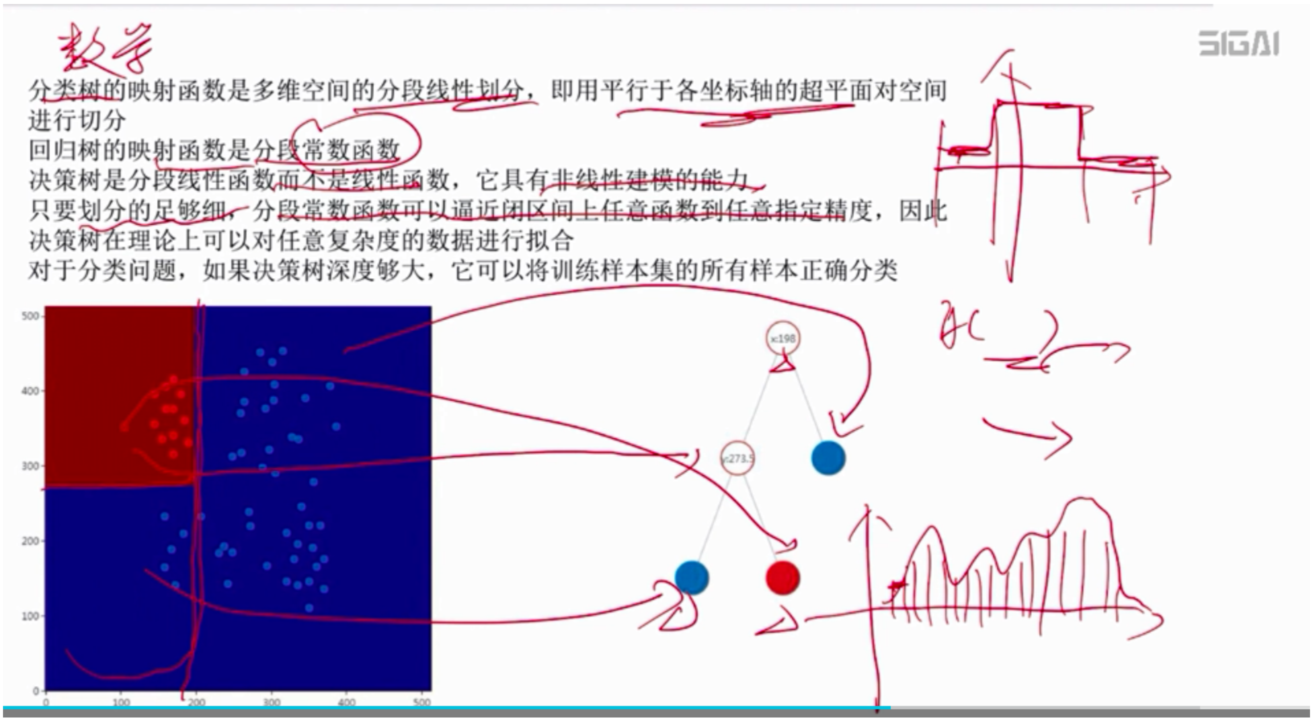
- 银行判断是否方法贷款
- 医生判断病人是否生病

在机器学习算法里面，



典型的决策树，分类与回归树。

6.4 决策树的表示能力



6.5 训练算法要解决的核心问题

训练算法

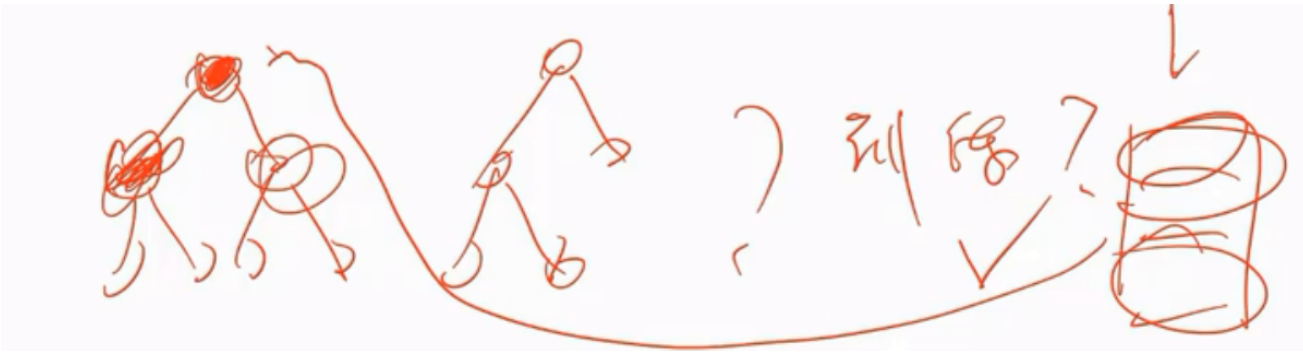
每个决策节点上应该选择哪个分量做判定？这个判定会将训练样本集一分为二，然后用这两个子集构造左右子树。

判定的规则是什么？即满足什么条件时进入左子树分支。对数值型变量要寻找一个分裂阈值进行判断，小于该阈值进入左子树，否则进入右子树。对于类别型变量则需要为它确定一个子集划分，将特征的取值集合划分成两个不相交的子集，如果特征的值属于第一个子集则进入左子树，否则进入右子树

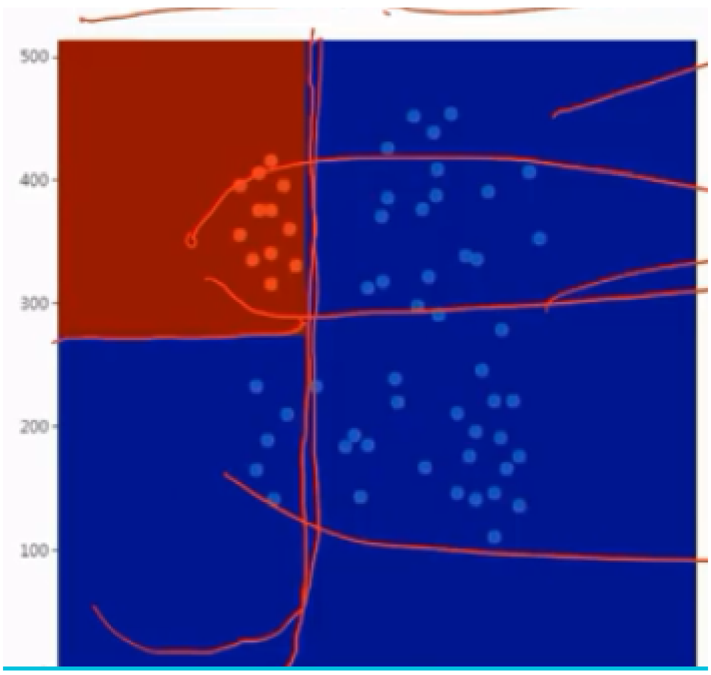
何时停止分裂，把节点设置为叶子节点？对于分类问题，当节点的样本都属于同一类型时停止，但这样可能会导致树的节点过多、深度过大，产生过拟合问题。另一种方法是当节点中的样本数小于一个阀值时停止分裂

如何为每个叶节点赋予类别标签或者回归值？即到达叶子节点时样本被分为哪一类或者赋予一个什么实数值

下图为一个决策树的例子



选择哪个特征分量做判定？



如果到达叶子节点。

决策树训练算法的核心。

6.6 递归分裂过程

递归分裂过程

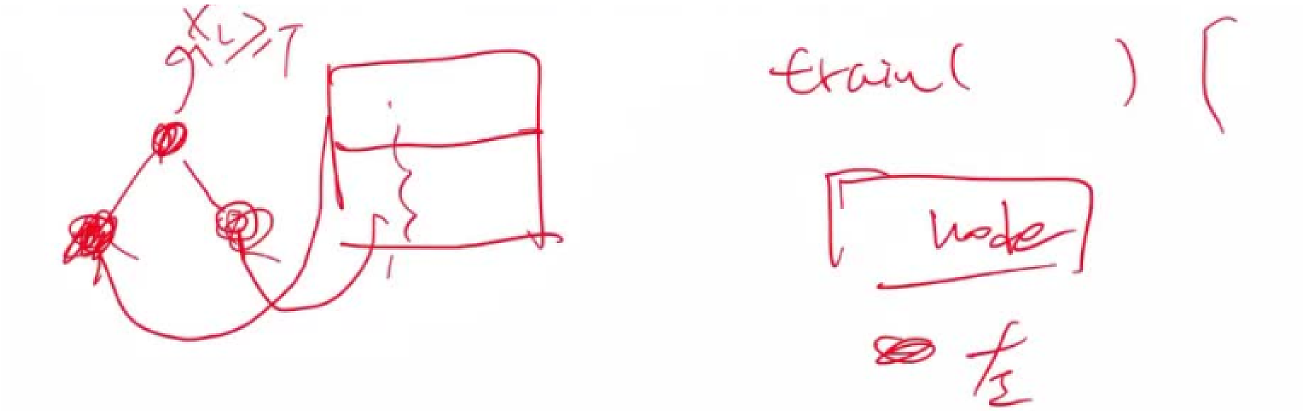
1.用样本集建立根节点，找到一个判定规则，将样本集D分裂成D1和D2两部分，同时为根节点设置判定规则

2.用样本集D1递归建立左子树

3.用样本集D2递归建立右子树

4.如果不能再进行分裂，则把节点标记为叶子节点，同时为它赋值

执行过程和代码如下图，左边为递归分裂过程，右边为代码实现：



6.7 寻找最佳分裂

整个决策树的训练过程是一个**递归的过程**。

1. x_2 ?
2.

寻找最佳分裂

对于分类问题，要保证分裂之后左右子树的样本尽可能的纯，即它们的样本尽可能属于不相交的某一类或者几类。为此需要定义不纯度的指标：当样本都属于某一类时不纯度为0；当样本均匀的属于所有类时不纯度最大

$$p_i = N_i / N$$

熵不纯度 $E(D) = -\sum_i p_i \log_2 p_i$

Gini不纯度 $G(D) = 1 - \sum_i p_i^2$ $\quad G(D) = 1 - \sum_i p_i^2 = 1 - \sum_i (N_i / N)^2 = 1 - \left(\sum_i N_i^2\right) / N^2$

误分类不纯度 $E(D) = 1 - \max(p_i)$

两个问题

1. x_2 ?

2. x_i — ? T ?

分类问题

这里只考虑数值型特征

pi=N/N

熵不纯度

$$\text{熵不纯度} \qquad E(D) = -\sum_i p_i \log_2 p_i$$

基尼系数不纯度

$$\text{Gini不纯度} \qquad G(D) = 1 - \sum_i p_i^2 \qquad G(D) = 1 - \sum_i p_i^2 = 1 - \sum_i (N_i / N)^2 = 1 - \left(\sum_i N_i^2 \right) / N^2$$

$$\text{误分类不纯度} \quad E(D) = 1 - \max(p_i)$$

第二部分

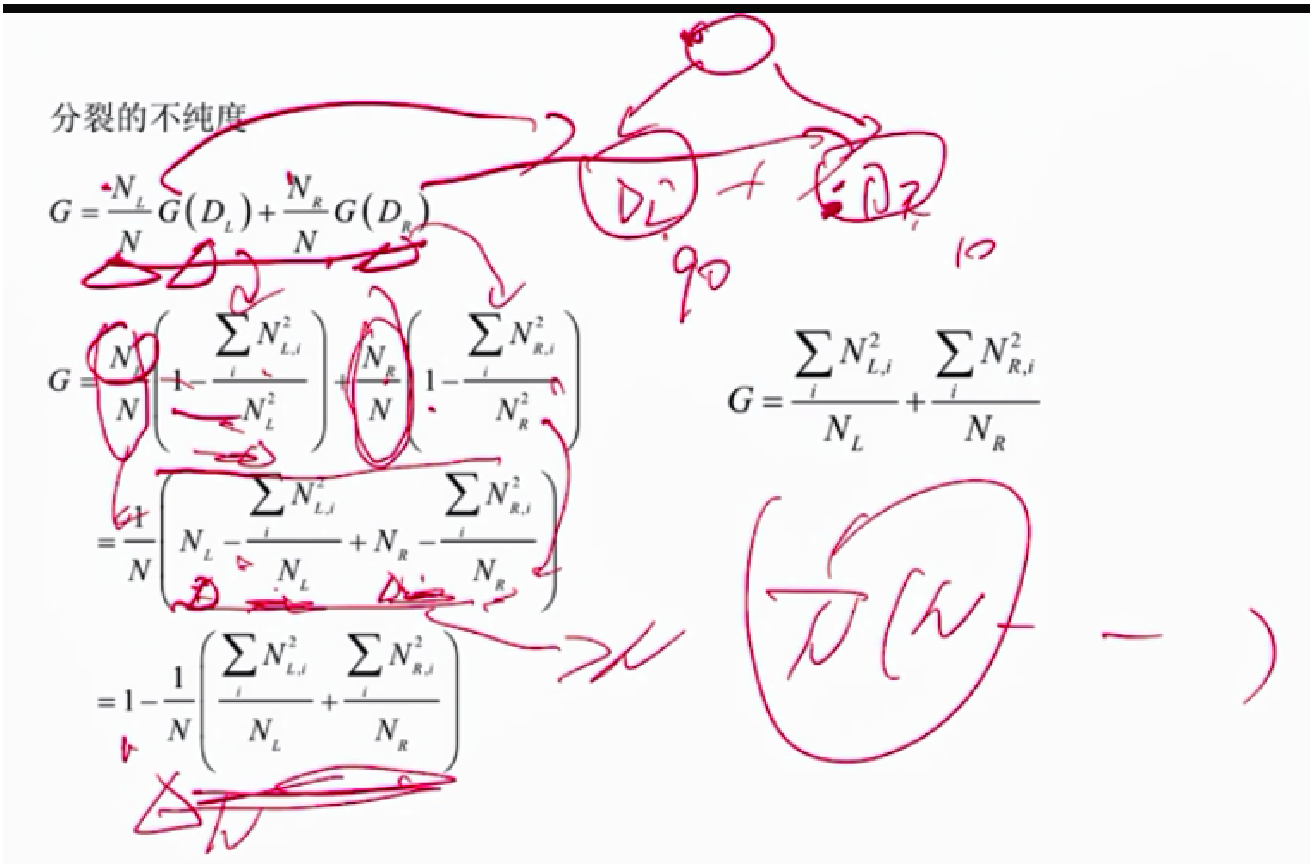
$$p_i = N_i / N$$

$$\text{熵不纯度} \qquad E(D) = -\sum_i p_i \log_2 p_i$$

$$\text{Gini不纯度} \qquad G(D) = 1 - \sum_i p_i^2 \qquad G(D) = 1 - \sum_i p_i^2 = 1 - \sum_i (N_i / N)^2 = 1 - \left(\sum_i N_i^2 \right) / N^2$$

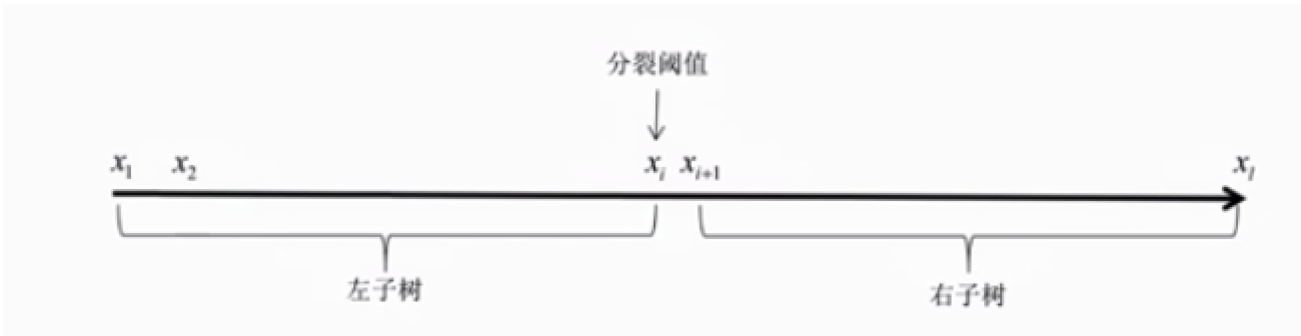
$$\text{误分类不纯度} \qquad E(D) = 1 - \max(p_i)$$

定义一个样本集



先将x排序，然后按照从小到大。

找出基尼纯度的最大值作为划分



回归问题

回归问题

$$E(D)=\frac{1}{l}\sum_{i=1}^{l}\left(y_i-\bar{y}\right)^2$$

$$\begin{aligned}E(D)&=\frac{1}{l}\sum_{i=1}^{l}\left(y_i-\frac{1}{l}\sum_{i=1}^{l}y_i\right)^2\\&=\frac{1}{l}\sum_{i=1}^{l}\left(y_i^2-2y_i\frac{1}{l}\sum_{i=1}^{l}y_i+\frac{1}{l^2}\left(\sum_{i=1}^{l}y_i\right)^2\right)\\&=\frac{1}{l}\left(\sum_{i=1}^{l}y_i^2-\frac{2}{l}\left(\sum_{i=1}^{l}y_i\right)^2+\frac{1}{l}\left(\sum_{i=1}^{l}y_i\right)^2\right)\\&=\frac{1}{l}\left(\sum_{i=1}^{l}y_i^2-\frac{1}{l}\left(\sum_{i=1}^{l}y_i\right)^2\right)\end{aligned}$$

$$E=E(D)-\frac{N_L}{N}E(D_L)-\frac{N_R}{N}E(D_R)$$

$$\begin{aligned}E&=\frac{1}{N}\left(\sum_{i=1}^{N}y_i^2-\frac{1}{N}\left(\sum_{i=1}^{N}y_i\right)^2\right)-\frac{N_L}{N}\left(\frac{1}{N_L}\left(\sum_{i=1}^{N_L}y_i^2-\frac{1}{N_L}\left(\sum_{i=1}^{N_L}y_i\right)^2\right)\right)-\\&\quad\frac{N_R}{N}\left(\frac{1}{N_R}\left(\sum_{i=1}^{N_R}y_i^2-\frac{1}{N_R}\left(\sum_{i=1}^{N_R}y_i\right)^2\right)\right)\\&=-\frac{1}{N^2}\left(\sum_{i=1}^{N}y_i\right)^2+\frac{1}{N}\left(\frac{1}{N_L}\left(\sum_{i=1}^{N_L}y_i\right)^2+\frac{1}{N_R}\left(\sum_{i=1}^{N_R}y_i\right)^2\right)\end{aligned}$$

$$E=\frac{1}{N_L}\left(\sum_{i=1}^{N_L}y_i\right)^2+\frac{1}{N_R}\left(\sum_{i=1}^{N_R}y_i\right)^2$$

回归误差

旧 - 新

最大分裂

决策树里的回归问题，真奇怪。

6.8 叶子节点值的设定

分为两种情况，对于分类树和对于回归树

叶子节点值的设定
如果不能继续分裂，则将该节点设置为叶子节点
对于分类树，将叶子节点的值设置成本节点的训练样本集中出现概率最大的那个类
对于回归树，则设置为本节点训练样本标签值的均值

6.9 属性缺失与替代分裂

属性缺失问题
在某些情况下样本特征向量中一些分量没有值，这称为属性缺失
对于每个决策树节点除了计算出一个最佳分裂规则作为主分裂规则，还会生成一个或者多个替代分裂规则作为备选。在预测时如果主分裂规则对应的特征出现缺失，则使用替代分裂规则进行判定。需要注意的是，替代分裂对于分类问题和回归问题是做相同的处理

$$\mathrm{LL}, \mathrm{LR}, \mathrm{RL}, \mathrm{RR}$$

$$\max\left(\mathrm{LL}+\mathrm{RR}, \mathrm{LR}+\mathrm{RL}\right)$$

属性缺失

属性缺失和为零是不一样。

主分裂规则

替代分裂规则，尽可能朱分裂规则类似

$$\mathrm{LL}, \mathrm{LR}, \mathrm{RL}, \mathrm{RR}$$

$$\max\left(\mathrm{LL}+\mathrm{RR}, \mathrm{LR}+\mathrm{RL}\right)$$

6.10 过拟合与剪枝

决策树是一种机器学习算法，它也有过拟合。

解决过拟合问题，有正则化，剪枝。

剪枝算法

$$\alpha = \frac{E(n) - E(n_t)}{|n_t| - 1}$$

$$E(n) = \frac{N - \max(N_i)}{N}$$

$$E(n) = \frac{1}{N}\left(\sum_i (y_i^2) - \frac{1}{N}\left(\sum_i y_i\right)^2\right)$$

$$T_0, \ldots, T_m$$

在训练过程中剪枝，叫做预剪枝，

在训练过程后剪枝，叫做后剪枝。有一种算法叫做CCP。这个剪枝，定义一个系数n， 以它为跟的子树。

alpha 错误率的上升率。

6.11 实验环节

\1. 决策树原理简介

决策树是一种基于规则的方法，属于判别模型，通常设计成二叉树。理解决策树的关键点在于决策树的生成和剪枝。决策树的典型实现方法有ID3、C4.5和CART。

下面介绍决策树CART：

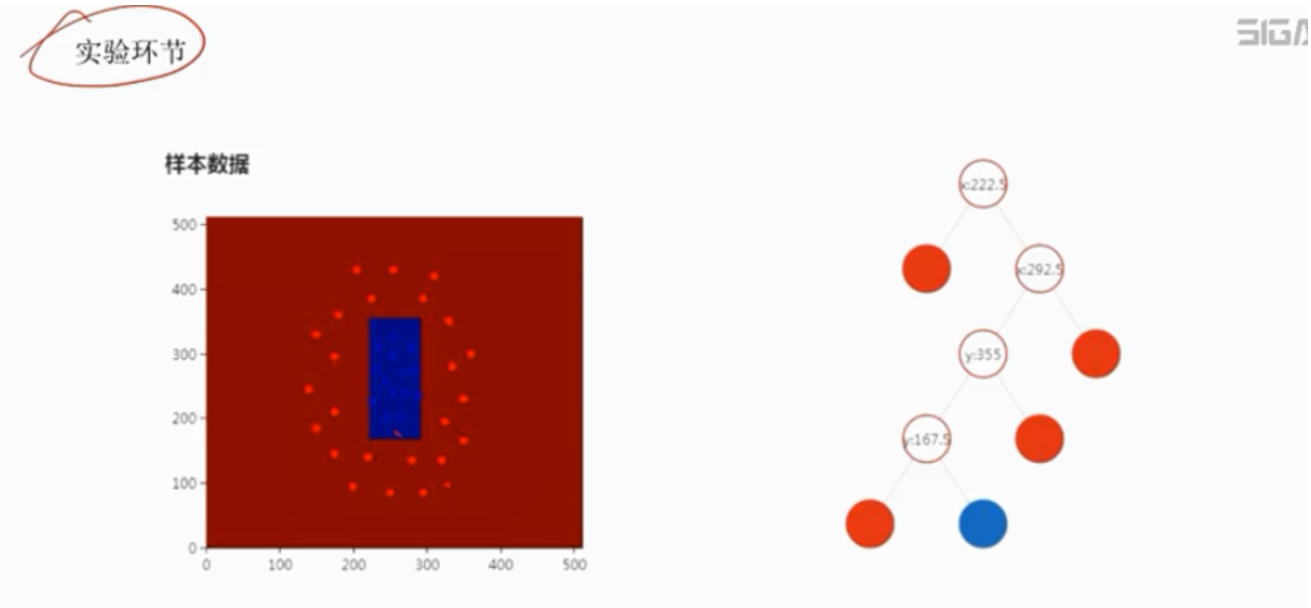
1）决策树的生成

CART的生成是递归构建二叉树的过程。对于分类树，在每个决策节点，CART采用最小化Gini不纯度来寻找最佳分裂。遍历每个特征以及每个特征的分割点，选择使划分后的样本Gini不纯度和最小的特征和分割点划分出两个子节点。如果训练样本数小于指定阈值或者树的深度达到最大深度，则不再进行分裂。

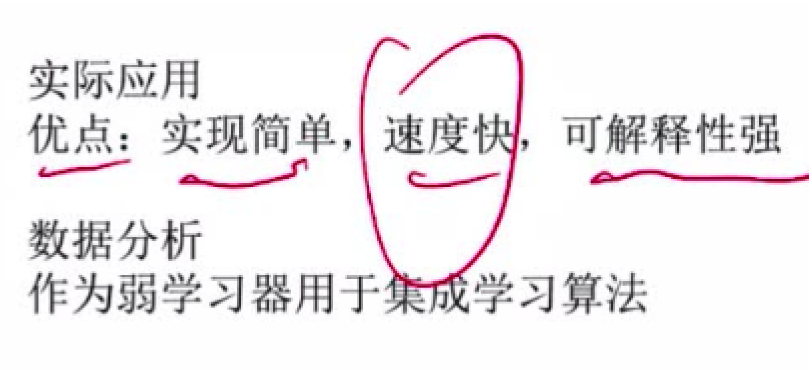
2）决策树的剪枝

如果决策树的结构太复杂，容易导致过拟合，此时需要进行剪枝。CART剪枝分两步：第一步用代价-复杂度剪枝算法逐步剪掉树的所有非叶子节点，直到只剩下根节点得到剪枝后的树序列；第二步根据真实误差值从上面的树序列中挑选出一棵树作为剪枝后的结果。

实验在云端实验室



6.12 实际应用



用于金融数据分析，医学数据分析。

6.13 决策树总结

回忆决策树的递归分裂过程。

课时总结：

这节课的东西多，概念也多。

重点：核心算法

难点：

疑点：寻找最佳分裂

个人自查自纠：

决策树是个重点算法

https://blog.csdn.net/ruggier/article/details/78756447

这篇博客将决策树很好，使用python来实现，不能做就看。

https://blog.csdn.net/jiaoyangwm/article/details/79525237#351-%E4%BD%BF%E7%94%A8sklearn%E6%9E%84%E5%BB%BA%E5%86%B3%E7%AD%96%E6%A0%91

实现代码

```
from math import log
import operator
from matplotlib.font_manager import FontProperties
import matplotlib.pyplot as plt
"""
函数说明：计算给定数据集的经验熵（香农熵）
Parameters：
    dataSet：数据集
Returns：
    shannonEnt：经验熵
```

```
Modify：
    2018-03-12

"""
def calcShannonEnt(dataSet):
    #返回数据集行数
    numEntries=len(dataSet)
    #保存每个标签（label）出现次数的字典
    labelCounts={}
    #对每组特征向量进行统计
    for featVec in dataSet:
        currentLabel=featVec[-1]                     #提取标签信息
        if currentLabel not in labelCounts.keys():   #如果标签没有放入统计次数的字典，添加进去
            labelCounts[currentLabel]=0
        labelCounts[currentLabel]+=1                 #label计数

    shannonEnt=0.0                                   #经验熵
    #计算经验熵
    for key in labelCounts:
        prob=float(labelCounts[key])/numEntries      #选择该标签的概率
        shannonEnt-=prob*log(prob,2)                 #利用公式计算
    return shannonEnt                                #返回经验熵

"""
函数说明：创建测试数据集
Parameters：无
Returns：
    dataSet：数据集
    labels：分类属性
Modify：
    2018-03-13

"""
def createDataSet():
    # 数据集
    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']]
    #分类属性
    labels=['年龄','有工作','有自己的房子','信贷情况']
    #返回数据集和分类属性
```

```
    return dataSet,labels

"""
函数说明：按照给定特征划分数据集

Parameters：
    dataSet:待划分的数据集
    axis：划分数据集的特征
    value：需要返回的特征值
Returns：
    无
Modify：
    2018-03-13

"""
def splitDataSet(dataSet,axis,value):
    #创建返回的数据集列表
    retDataSet=[]
    #遍历数据集
    for featVec in dataSet:
        if featVec[axis]==value:
            #去掉axis特征
            reduceFeatVec=featVec[:axis]
            #将符合条件的添加到返回的数据集
            reduceFeatVec.extend(featVec[axis+1:])
            retDataSet.append(reduceFeatVec)
    #返回划分后的数据集
    return retDataSet

"""
函数说明：计算给定数据集的经验熵（香农熵）
Parameters：
    dataSet：数据集
Returns：
    shannonEnt：信息增益最大特征的索引值
Modify：
    2018-03-13

"""



def chooseBestFeatureToSplit(dataSet):
    #特征数量
    numFeatures = len(dataSet[0]) - 1
    #计数数据集的香农熵
    baseEntropy = calcShannonEnt(dataSet)
    #信息增益
    bestInfoGain = 0.0
    #最优特征的索引值
    bestFeature = -1
    #遍历所有特征
    for i in range(numFeatures):
        # 获取dataSet的第i个所有特征
```

```
        featList = [example[i] for example in dataSet]
        #创建set集合{},元素不可重复
        uniqueVals = set(featList)
        #经验条件熵
        newEntropy = 0.0
        #计算信息增益
        for value in uniqueVals:
            #subDataSet划分后的子集
            subDataSet = splitDataSet(dataSet, i, value)
            #计算子集的概率
            prob = len(subDataSet) / float(len(dataSet))
            #根据公式计算经验条件熵
            newEntropy += prob * calcShannonEnt((subDataSet))
        #信息增益
        infoGain = baseEntropy - newEntropy
        #打印每个特征的信息增益
        print("第%d个特征的增益为%.3f" % (i, infoGain))
        #计算信息增益
        if (infoGain > bestInfoGain):
            #更新信息增益,找到最大的信息增益
            bestInfoGain = infoGain
            #记录信息增益最大的特征的索引值
            bestFeature = i
            #返回信息增益最大特征的索引值
    return bestFeature

"""
函数说明:统计classList中出现次数最多的元素(类标签)
Parameters:
    classList:类标签列表
Returns:
    sortedClassCount[0][0]:出现次数最多的元素(类标签)
Modify:
    2018-03-13

"""
def majorityCnt(classList):
    classCount={}
    #统计classList中每个元素出现的次数
    for vote in classList:
        if vote not in classCount.keys():
            classCount[vote]=0
            classCount[vote]+=1
        #根据字典的值降序排列

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

"""
函数说明:创建决策树

Parameters:
    dataSet:训练数据集
```

```
    labels：分类属性标签
    featLabels：存储选择的最优特征标签
Returns：
    myTree：决策树
Modify：
    2018-03-13

"""
def createTree(dataSet,labels,featLabels):
    #取分类标签（是否放贷：yes or no）
    classList=[example[-1] for example in dataSet]
    #如果类别完全相同，则停止继续划分
    if classList.count(classList[0])==len(classList):
        return classList[0]
    #遍历完所有特征时返回出现次数最多的类标签
    if len(dataSet[0])==1:
        return majorityCnt(classList)
    #选择最优特征
    bestFeat=chooseBestFeatureToSplit(dataSet)
    #最优特征的标签
    bestFeatLabel=labels[bestFeat]
    featLabels.append(bestFeatLabel)
    #根据最优特征的标签生成树
    myTree={bestFeatLabel:{}}
    #删除已经使用的特征标签
    del(labels[bestFeat])
    #得到训练集中所有最优特征的属性值
    featValues=[example[bestFeat] for example in dataSet]
    #去掉重复的属性值
    uniqueVls=set(featValues)
    #遍历特征，创建决策树
    for value in uniqueVls:
        myTree[bestFeatLabel][value]=createTree(splitDataSet(dataSet,bestFeat,value),
                                               labels,featLabels)
    return myTree

"""
函数说明：获取决策树叶子节点的数目

Parameters：
    myTree：决策树
Returns：
    numLeafs：决策树的叶子节点的数目
Modify：
    2018-03-13

"""

def getNumLeafs(myTree):
    numLeafs=0
    firstStr=next(iter(myTree))
    secondDict=myTree[firstStr]
    for key in secondDict.keys():
```

```
        if type(secondDict[key]).__name__=='dict':
            numLeafs+=getNumLeafs(secondDict[key])
        else: numLeafs+=1
    return numLeafs

"""
函数说明:获取决策树的层数

Parameters:
    myTree:决策树
Returns:
    maxDepth:决策树的层数

Modify:
    2018-03-13
"""
def getTreeDepth(myTree):
    maxDepth = 0                                                #初始化决策树深度
    firstStr = next(iter(myTree))                               #python3中
myTree.keys()返回的是dict_keys,不在是list,所以不能使用myTree.keys()[0]的方法获取结点属性，可以使
用list(myTree.keys())[0]
    secondDict = myTree[firstStr]                               #获取下一个字典
    for key in secondDict.keys():
        if type(secondDict[key]).__name__=='dict':                #测试该结点是否为字典，如
果不是字典，代表此结点为叶子结点
            thisDepth = 1 + getTreeDepth(secondDict[key])
        else:   thisDepth = 1
        if thisDepth > maxDepth: maxDepth = thisDepth           #更新层数
    return maxDepth

"""
函数说明:绘制结点

Parameters:
    nodeTxt - 结点名
    centerPt - 文本位置
    parentPt - 标注的箭头位置
    nodeType - 结点格式
Returns:
    无
Modify:
    2018-03-13
"""
def plotNode(nodeTxt, centerPt, parentPt, nodeType):
    arrow_args = dict(arrowstyle="<-")                                            #定义
箭头格式
    font = FontProperties(fname=r"c:\windows\fonts\simsun.ttc", size=14)        #设置中
文字体
    createPlot.ax1.annotate(nodeTxt, xy=parentPt,  xycoords='axes fraction',    #绘制结
点
        xytext=centerPt, textcoords='axes fraction',
        va="center", ha="center", bbox=nodeType, arrowprops=arrow_args,
FontProperties=font)
```

```
"""
函数说明:标注有向边属性值

Parameters:
    cntrPt、parentPt - 用于计算标注位置
    txtString - 标注的内容
Returns:
    无
Modify:
    2018-03-13
"""
def plotMidText(cntrPt, parentPt, txtString):
    xMid = (parentPt[0]-cntrPt[0])/2.0 + cntrPt[0]
       #计算标注位置
    yMid = (parentPt[1]-cntrPt[1])/2.0 + cntrPt[1]
    createPlot.ax1.text(xMid, yMid, txtString, va="center", ha="center", rotation=30)

"""
函数说明:绘制决策树

Parameters:
    myTree - 决策树(字典)
    parentPt - 标注的内容
    nodeTxt - 结点名
Returns:
    无
Modify:
    2018-03-13
"""
def plotTree(myTree, parentPt, nodeTxt):
    decisionNode = dict(boxstyle="sawtooth", fc="0.8")
       #设置结点格式
    leafNode = dict(boxstyle="round4", fc="0.8")
     #设置叶结点格式
    numLeafs = getNumLeafs(myTree)
     #获取决策树叶结点数目，决定了树的宽度
    depth = getTreeDepth(myTree)
     #获取决策树层数
    firstStr = next(iter(myTree))
     #下个字典
    cntrPt = (plotTree.xOff + (1.0 + float(numLeafs))/2.0/plotTree.totalW,
plotTree.yOff)    #中心位置
    plotMidText(cntrPt, parentPt, nodeTxt)
       #标注有向边属性值
    plotNode(firstStr, cntrPt, parentPt, decisionNode)
       #绘制结点
    secondDict = myTree[firstStr]
     #下一个字典，也就是继续绘制子结点
    plotTree.yOff = plotTree.yOff - 1.0/plotTree.totalD
       #y偏移
    for key in secondDict.keys():
```

```
        if type(secondDict[key]).__name__=='dict':
       #测试该结点是否为字典，如果不是字典，代表此结点为叶子结点
            plotTree(secondDict[key],cntrPt,str(key))
     #不是叶结点，递归调用继续绘制
        else:
     #如果是叶结点，绘制叶结点，并标注有向边属性值
            plotTree.xOff = plotTree.xOff + 1.0/plotTree.totalW
            plotNode(secondDict[key], (plotTree.xOff, plotTree.yOff), cntrPt, leafNode)
            plotMidText((plotTree.xOff, plotTree.yOff), cntrPt, str(key))
    plotTree.yOff = plotTree.yOff + 1.0/plotTree.totalD

"""
函数说明:创建绘制面板

Parameters:
    inTree - 决策树(字典)
Returns:
    无
Modify:
    2018-03-13
"""
def createPlot(inTree):
    fig = plt.figure(1, facecolor='white')#创建fig
    fig.clf()#清空fig
    axprops = dict(xticks=[], yticks=[])
    createPlot.ax1 = plt.subplot(111, frameon=False, **axprops)#去掉x、y轴
    plotTree.totalW = float(getNumLeafs(inTree))#获取决策树叶结点数目
    plotTree.totalD = float(getTreeDepth(inTree))#获取决策树层数
    plotTree.xOff = -0.5/plotTree.totalW; plotTree.yOff = 1.0#x偏移
    plotTree(inTree, (0.5,1.0), '')#绘制决策树
    plt.show()#显示绘制结果

if __name__ == '__main__':
    dataSet, labels = createDataSet()
    featLabels = []
    myTree = createTree(dataSet, labels, featLabels)
    print(myTree)
    createPlot(myTree)

if __name__=='__main__':
    dataSet,labels=createDataSet()
    featLabels=[]
    myTree=createTree(dataSet,labels,featLabels)
    print(myTree)
```