# 决策树与集成学习

信息学院-黄浩

2022年3月14日星期一

# 主要内容

一、信息论基本概念

二、决策树基础

三、集成学习

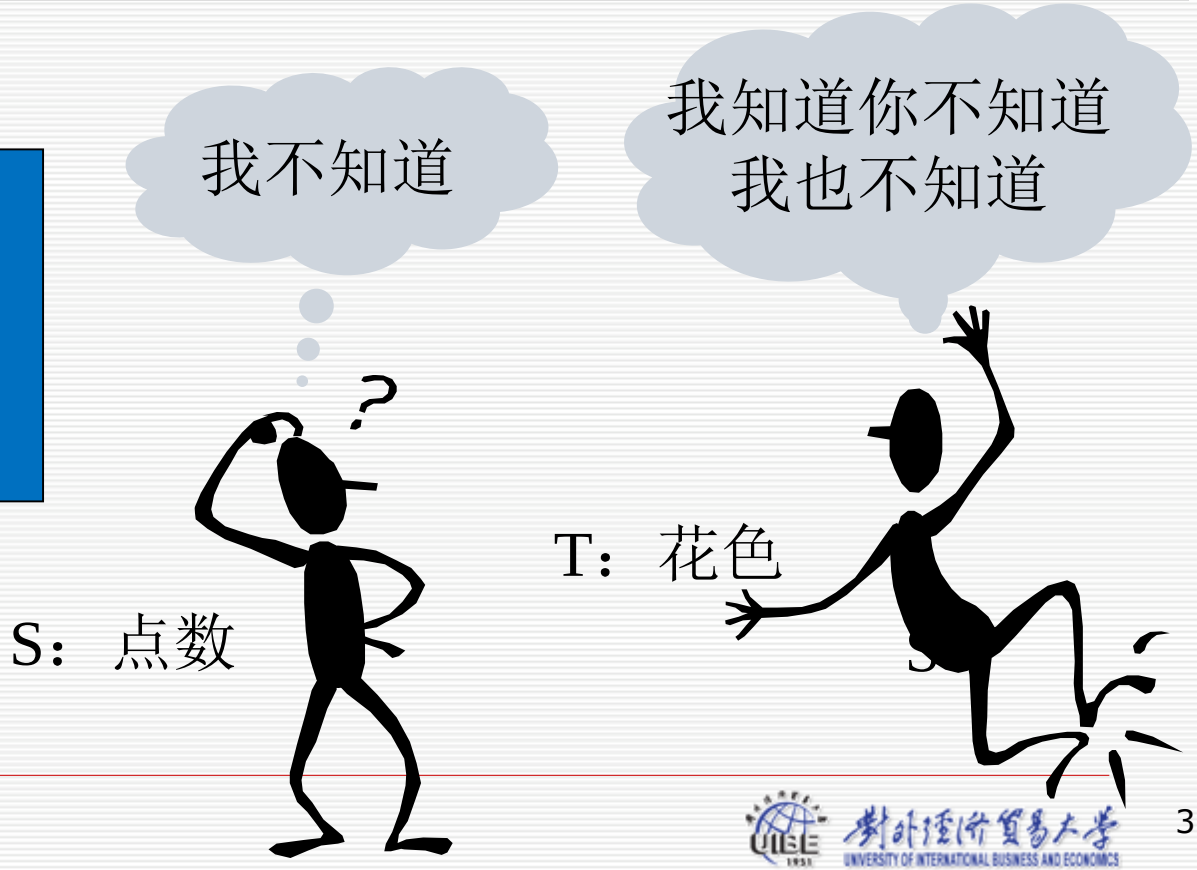
四、随机森林

五、GBDT与XGBoost

# 基本概念

## S先生和T先生的故事

黑桃: A 10 8 4 2
红桃: A 5 4
方块: Q 5
梅花: K Q 3



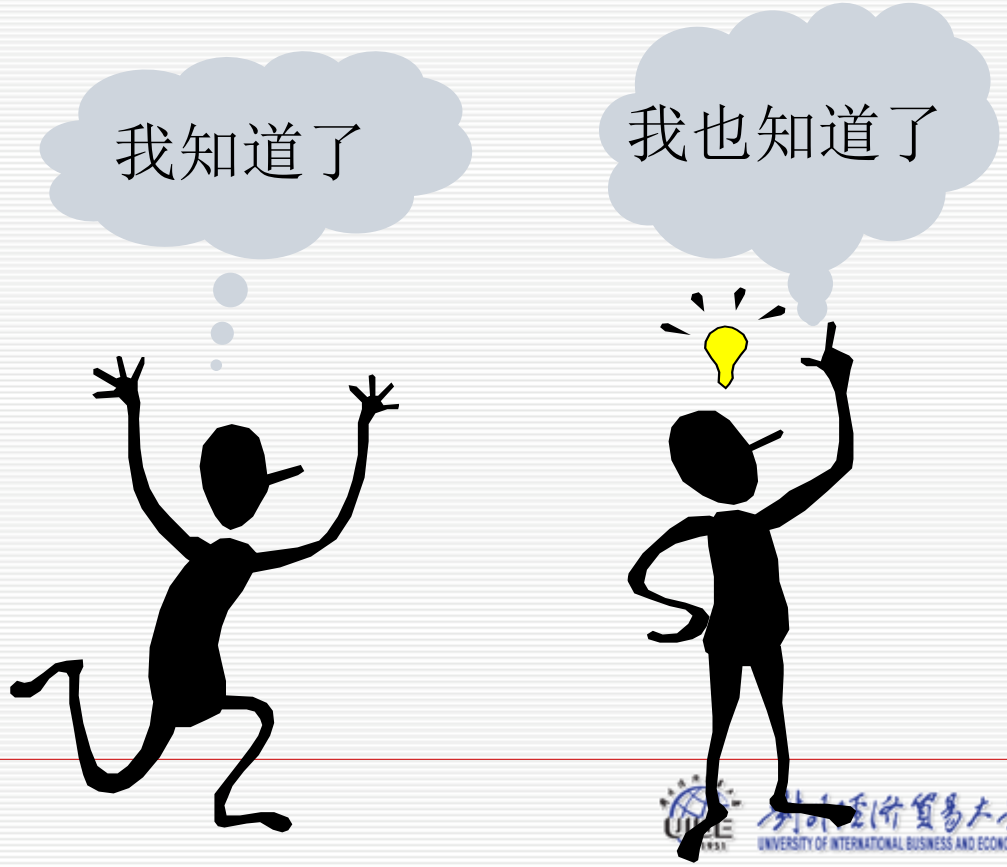
# 基本概念

## S先生和T先生的故事

黑桃: A 10 8 4 2
红桃: A 5 4
方块: Q 5
梅花: K Q 3

你知道了吗?

方块Q!



# 基本概念

- S先生:"我不知道这张牌" =>点数为4、5、A、Q
- T先生:"我知道你不知道这张牌" =>花色是红桃或方块
- S先生:"现在我知道这张牌了" =>点数为4、A、Q
- T先生:"我也知道了" =>方块Q
  如果不是方块Q，T先生不可能知道，因为红桃4和红桃A都有可能

# 基本概念

信息的要害是改变知识状态

# 基本概念

1、箱子里面有橙、紫、蓝及青四种颜色的小球任意个，各颜色小球的占比相同，现在从中拿出一个小球，猜小球是什么颜色？

- 1/4概率是橙色球，需要猜两次，1/4是紫色球，需要猜两次，其余的小球类似，所以预期的猜球次数为：
- H = 1/4 * 2 + 1/4 * 2 + 1/4 * 2 + 1/4 * 2 = 2

# 基本概念

2、箱子里面有小球任意个，其中1/2是橙色球，1/4是紫色球，1/8是蓝色球及1/8是青色球。从中拿出一个球，猜球是什么颜色的？

- 橙色占比二分之一，如果猜橙色，可能第一次就猜中了。1/2的概率是橙色球，需要猜一次，1/4的概率是紫色球，需要猜两次，1/8的概率是蓝色球，需要猜三次，1/8的概率是青色球，需要猜三次，所以预期的猜题次数为：
- H = 1/2 * 1 + 1/4 * 2 + 1/8 * 3 + 1/8 * 3= 1.75

# 基本概念

3、箱子里面的球都是橙色，从中拿出一个，猜球是什么颜色？

- 需要猜0次。

上面三个题目表现出这样一种现象：针对特定概率为p的小球，需要猜球的次数 = $\log_2 \frac{1}{p}$，例如题目2中，1/4是紫色球，$\log_2 4 = 2$次，1/8是蓝色球，$\log_2 8 = 3$次。那么，针对整个整体，预期的猜题次数为：$\sum_{k=1}^{N} p_k \log_2 \frac{1}{p_k}$，这就是**信息熵**，上面三个题目的预期猜球次数都是由这个公式计算而来，第一题的信息熵为2，第二题的信息熵为1.75，最三题的信息熵为$1 * \log 1 = 0$。

# 基本概念

**信息熵代表的是随机变量或整个系统的不确定性，熵越大，随机变量或系统的不确定性就越大。**

- 在题目1中，整个系统所有的情况出现的概率都是均等的，此时的熵是最大的。
- 题目2中，知道了橙色小球出现的概率是1/2及其他小球各自出现的概率，说明对这个系统有一定的了解，所以系统的不确定性自然会降低，所以熵小于2。
- 题目3中，已经知道箱子中肯定是橙色球，球肯定是橙色的，因而整个系统的不确定性为0，也就是熵为0。

# 基本概念

根据真实分布，我们能够找到一个最优策略，以最小的代价消除系统的不确定性，而这个代价大小就是信息熵。

信息熵衡量了系统的不确定性，而我们要消除这个不确定性，所要付出的【最小努力】（猜题次数、编码长度等）的大小就是信息熵。

# 基本概念

热力学中的热熵是表示分子状态混乱程度的物理量。 Shannon用信息熵的概念来描述信源的不确定度。

Shannon 借鉴了热力学的概念，把信息中排除了冗余后的平均信息量称为“信息熵”，并给出了计算信息熵的数学表达式。

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

联合熵 $$H(XY) = \sum_{x,y} p(xy) I(xy) = \sum_{x,y} p(xy) \log \frac{1}{p(xy)}$$

# 基本概念

- H(X,Y) – H(X)
  - (X,Y)发生所包含的熵，减去X单独发生包含的熵：在X发生的前提下，Y发生“新”带来的熵
  - 该式子定义为X发生前提下，Y的熵：
    - 条件熵H(Y|X)

# 基本概念

$$
\begin{aligned}
& H(X, Y) - H(X) \\
&= -\sum_{x,y} p(x, y) \log\, p(x, y) + \sum_{x} p(x) \log\, p(x) \\
&= -\sum_{x,y} p(x, y) \log\, p(x, y) + \sum_{x} \left( \sum_{y} p(x, y) \right) \log\, p(x) \\
&= -\sum_{x,y} p(x, y) \log\, p(x, y) + \sum_{x,y} p(x, y) \log\, p(x) \\
&= -\sum_{x,y} p(x, y) \log \frac{p(x, y)}{p(x)} \\
&= -\sum_{x,y} p(x, y) \log\, p(y \mid x)
\end{aligned}
$$

# 基本概念

$$
\begin{aligned}
H(X,Y) - H(X) &= -\sum_{x,y} p(x,y)\log p(y \mid x) \\
&= -\sum_{x}\sum_{y} p(x,y)\log p(y \mid x) \\
&= -\sum_{x}\sum_{y} p(x)p(y \mid x)\log p(y \mid x) \\
&= -\sum_{x} p(x)\sum_{y} p(y \mid x)\log p(y \mid x) \\
&= \sum_{x} p(x)\left(-\sum_{y} p(y \mid x)\log p(y \mid x)\right) \\
&= \sum_{x} p(x)H(Y \mid X = x)
\end{aligned}
$$

# 基本概念

交叉熵：

假设小球的真实分布是（1/2，1/4, 1/8，1/8），但所选择的策略却认为所有的小球出现的概率相同，相当于忽略了箱子中各小球的**真实分布**，而仍旧认为所有小球出现的概率是一样的，认为小球的分布为（1/4，1/4，1/4，1/4），这个分布就是**非真实分布**。此时，猜中任何一种颜色的小球都需要猜两次，即1/2 * 2 + 1/4 * 2 + 1/8 * 2 + 1/8 * 2 = 2。

**在给定的真实分布下，使用非真实分布所指定的策略消除系统的不确定性所需要付出的努力的大小。**

交叉熵越低，这个策略就越好，最低的交叉熵也就是使用了真实分布所计算出来的信息熵，此时 ，交叉熵 = 信息熵。

# 基本概念

相对熵：又名互熵、鉴别信息、Kullback-Leible散度等。

$$
\begin{aligned}
D_{KL}(p||q) &= E_p[\log \frac{p(x)}{q(x)}] = \sum_{x\in\mathcal{X}} p(x)\log\frac{p(x)}{q(x)} \\
&= \sum_{x\in\mathcal{X}} [p(x)\log p(x) - p(x)\log q(x)] \\
&= \sum_{x\in\mathcal{X}} p(x)\log p(x) - \sum_{x\in\mathcal{X}} p(x)\log q(x) \\
&= -H(p) - \sum_{x\in\mathcal{X}} p(x)\log q(x) \\
&= -H(p) + E_p[-\log q(x)] \\
&= H_p(q) - H(p)
\end{aligned}
$$

# 基本概念

相对熵：又名互熵、鉴别信息、Kullback-Leible散度等。

相对熵具有如下性质：

- 相对熵可以度量两个随机变量的距离；
- 一般不具有对称性，即D(p||q) ≠ D(q||p)，当且仅当p = q, 则相对熵为0，二者相等；
- D(p||q) >= 0, D(q||p) >= 0.

那么，应该使用D(p||q) 还是 D(q||p)呢？

假定已知随机变量P，求一个随机变量Q，使得Q尽量接近于P，这样可以使用P和Q的K-L来度量他们的距离。

- 假定使用KL(Q || P)，为了让距离最小，则要求P为0的地方，Q尽量为0。这样会得到比较瘦高的分布曲线；
- 假定使用KL(P || Q)，为了让距离最小，则要求P不为0 的地方，Q也尽量不为0。这样会得到比较矮胖的分布曲线。

# 基本概念

交叉熵：

$$D_{KL}(p||q) = \sum_{i=1}^{n} p(x_i) log(p(x_i)) - \sum_{i=1}^{n} p(x_i) log(q(x_i))$$
$$= -H(p(x)) + [-\sum_{i=1}^{n} p(x_i) log(q(x_i))]$$

$$H(p,q) = -\sum_{i=1}^{n} p(x_i) log(q(x_i))$$

# 基本概念

在机器学习中，需要评估label和predicts之间的差距，使用KL散度刚刚好，即 $D_{KL}(y||\hat{y})$，由于KL散度中的前一部分 $-H(y)$ 不变，故在优化过程中，只需要关注交叉熵就可以了。所以一般在机器学习中直接用用交叉熵做损失函数，评估模型。

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

$$loss = -\sum_{i=1}^{n} y_i log(\hat{y}_i)$$

- 对于第一项Loss函数，如果是Square loss，那就是最小二乘了；
- 如果是Hinge Loss，那就是著名的SVM了；
- 如果是exp-Loss，那就是 Boosting；
- 如果是log-Loss，那就是Logistic Regression了；
- 还有很多。不同的loss函数，具有不同的拟合特性，需要看具体问题具体分析。

# 基本概念

互信息：两个随机变量X，Y的互信息，定义为X，Y的联合分布和独立分布乘积的相对熵。

即： I(X, Y) = D(P(X, Y) || P(X)P(Y))

$$I(X,Y)=\sum_{x,y}p(x,y)\log\frac{p(x,y)}{p(x)p(y)}$$

即：

可以通过简单的计算得到：

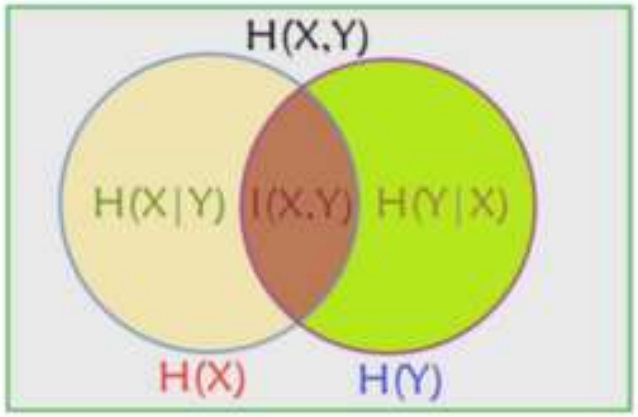
H(X|Y) = H(X) - I(X, Y),

互信息为0，则随机变量X和Y是互相独立的。

# 基本概念

- H(X|Y) = H(X, Y) - H(Y); H(Y|X) = H(X, Y) - H(X) —— 条件熵的定义
- H(X|Y) = H(X) - I(X, Y); H(Y|X) = H(Y) - I(X, Y)
- I(X, Y) = H(X) - H(X|Y) = H(X) + H(Y) - H(X, Y) —— 也可以作为互信息的定义
- H(X|Y) <= H(X):
  - H(X)表示X的不确定度；H(X|Y)表示给定Y的情况下，X的不确定度。
  - 如果X与Y完全独立，则二者相等（给不给Y对X能给出多少信息无关）；
  - 而如果X与Y不是独立的，则给定Y之后会降低X的熵，即X的不确定性会降低。

用Venn图帮助记忆：



# 基本概念-最大熵模型

最大熵模型的原则：

◆承认已知事物（知识）；

◆对未知事物不做任何假设，没有任何偏见。

对一个随机事件的概率分布进行预测时，预测应当满足全部已知条件，而对未知的情况不要做任何主观假设。在这种情况下，概率分布最均匀，预测的风险最小。

因为这时概率分布的信息熵最大，所以把这种模型叫做“最大熵模型”（Maximum Entropy）。

一般模型：

$$\max_{p \in P} H(Y \mid X) = -\sum_{(x,y)} p(x,y) \log p(y \mid x)$$

P={p | p是X上满足条件的概率分布}

# 基本概念-最大熵模型

含$\lambda_i$的第一个约束项表示模型要能够很好的解释初始数据集，$f_i(x, y)$表示选取的第i个特征；含$v_0$的第二个约束项表示概率加和为1.
p(x, y) = p(y | x) * p(x)，而p(x)是已知的，所以用p(x)_bar来表示已知量。

$$L=\left(-\sum_{(x,y)}p(y|x)\bar{p}(x)\log p(y|x)\right)$$
$$+\left(\sum_i\lambda_i\sum_{(x,y)}f_i(x,y)\left[p(y|x)\bar{p}(x)-\bar{p}(x,y)\right]\right)+v_0\left[\sum_y p(y|x)-1\right]$$

# 基本概念-最大熵模型

对p(y | x)求偏导：

$$\frac{\partial L}{\partial p(y \mid x)} = \overline{p}(x)(-\log p(y \mid x) - 1) + \sum_i \lambda_i \overline{p}(x) f_i(x, y) + v_0 \stackrel{\Delta}{=} 0$$

$$\Rightarrow \left( \text{令} \lambda_0 = \frac{v_0}{\overline{p}(x)} \right) \Rightarrow$$

$$p^*(y \mid x) = \exp\left( \sum_i \lambda_i f_i(x, y) + \lambda_0 - 1 \right) = \frac{1}{\exp(1 - \lambda_0)} \cdot \exp\left( \sum_i \lambda_i f_i(x, y) \right)$$

# 基本概念-最大熵模型

归一化

$$p^*(y \mid x) = \frac{1}{Z_\lambda(x)} \exp\left(\sum_i \lambda_i f_i(x, y)\right)$$

$$\sum_y \frac{1}{Z_\lambda(x)} \exp\left(\sum_i \lambda_i f_i(x, y)\right) = 1 \Rightarrow Z_\lambda(x) = \sum_y \exp\left(\sum_i \lambda_i f_i(x, y)\right)$$

# 与Logistic/SoftMax回归的对比

Logistic/SoftMax回归的后验概率形式：

$$\begin{cases} h(c=1\mid x;\theta)=\dfrac{1}{1+e^{-\theta^T x}}=\dfrac{e^{\theta^T x}}{e^{\theta^T x}+1}\propto e^{\theta^T x} \\ h(c=0\mid x;\theta)=\dfrac{e^{-\theta^T x}}{1+e^{-\theta^T x}}=\dfrac{1}{e^{\theta^T x}+1}\propto 1 \end{cases}$$

$$h(c=k\mid x;\theta)=\frac{e^{\theta_k^T x}}{\sum_{j=1}^{K} e^{\theta_j^T x}},\quad k=1,2,\cdots,K$$

最大熵模型的后验概率形式：

$$p^*(y\mid x)=\frac{1}{Z_\lambda(x)}\exp\left(\sum_i \lambda_i f_i(x,y)\right)$$

# 与Logistic/SoftMax回归的对比

最大熵模型由最大熵原理推导出来，最大熵原理是概率模型学习或估计的一个准则，最大熵原理认为在所有可能的概率模型的集合中，熵最大的模型是最好的模型，最大熵模型也可以用于二类分类和多类分类。

Logistic回归模型与最大熵模型都属于对数线性模型。

逻辑回归跟最大熵模型**没有本质区别**。逻辑回归是最大熵对应类别为**二类**时的特殊情况。

**二项式**分布的最大熵解等价于二项式指数形式(**sigmoid**)的最大似然；
**多项式**分布的最大熵等价于多项式分布指数形式(**softmax**)的最大似然。

# 最小鉴别信息

设随机变量 X 具有未知的概率分布密度函数 P(x)，其对已知若干函数 $f_m(x)$ 的数学期望为

$$\int p(x) f_m(x) dx = c_m \quad m = 1,2,\cdots M \text{ (约束条件)}$$

问题：当已知先验概率分布密度 $p_1(x)$ 的条件下，如何对 p(x)作出估计？

因为有约束条件 $\int p(x) dx = 1$

如果以最小鉴别信息最小为准则，即

$$\min_{p(x)} I(P(x), p_1(x)) = \min_{p(x)} \int p(x) \log \frac{p(x)}{p_1(x)} dx$$

则可保证当概率密度函数从 $p_1(x)$ 到 $p(x)$ 改变量时需要的信息量最小，使鉴别信息最小的分布是满足约束条件下最接近 $p_1(x)$ 的概率分布。

# 最小鉴别信息

引入拉格朗日乘子求最小值。

设函数 $F=\sum P(a_i)\log\frac{p(a_i)}{p_1(a_i)}-\beta(p(a_i)-1)-\sum\lambda_m p(a_i)f_m(a_i)$

即：$F=\int\left(P(x)\log\frac{p(x)}{p_1(x)}-\beta p(x)-\sum_{m=1}^{M}\lambda_m p(x)f_m(x)\right)dx$

计算 F 的偏导数为零，相当于计算

$$\frac{\partial}{\partial p(x)}\left[P(x)\log\frac{p(x)}{p_1(x)}-\beta p(x)-\sum_{m=1}^{M}\lambda_m p(x)f_m(x)\right]=0$$

因此有 $\log\frac{p(x)}{p_1(x)}+1-\beta-\sum_{m=1}^{M}\lambda_m f_m(x)=0$

注：$$p(x)\frac{\partial\log\frac{p(x)}{p_1(x)}}{\partial p(x)}=p(x)\cdot\frac{p_1(x)}{p(x)}\cdot\frac{1}{p_1(x)}=1$$

解得：

$$p(x)=p_1(x)\exp\left[\lambda_0+\sum_{m=1}^{M}\lambda_m f_m(x)\right]\qquad \text{其中：}\ \lambda_0=\beta-1$$

# 最小鉴别信息与最大熵原理

一般最大熵是最小鉴别信息原理的特例。
最大熵原理主要应用于离散分布和对先验分布未知的情况，所以先验概率分布不出现在运算中。
最小鉴别信息原理和最大熵原理目标是在给定先验概率分布或未知先验概率分布及已知对概率分布具有若干约束条件下（M＋1个条件），对真实的概率分布给出一个最佳或最优估计。
最小鉴别信息原理和最大熵原理，对非概率密度函数问题和非统计问题仍然适用。

# 最小鉴别信息与最大熵原理

最小鉴别信息为 $\min_{p(x)} I(P(x), p_1(x)) = \min_{p(x)} \int p(x)\log\frac{p(x)}{p_1(x)}dx$

因为当 $p_1(x)$ 为等概率先验分布时有：

$$\min_{p(x)} I(p(x), p_1(x)) = \min_{p(x)} \int p(x)\log p(x)dx + \log(\frac{1}{p_1(x)})$$

$$= \min_{p(x)} [-\int -p(x)\log p(x)dx] + \log(\frac{1}{p_1(x)})$$

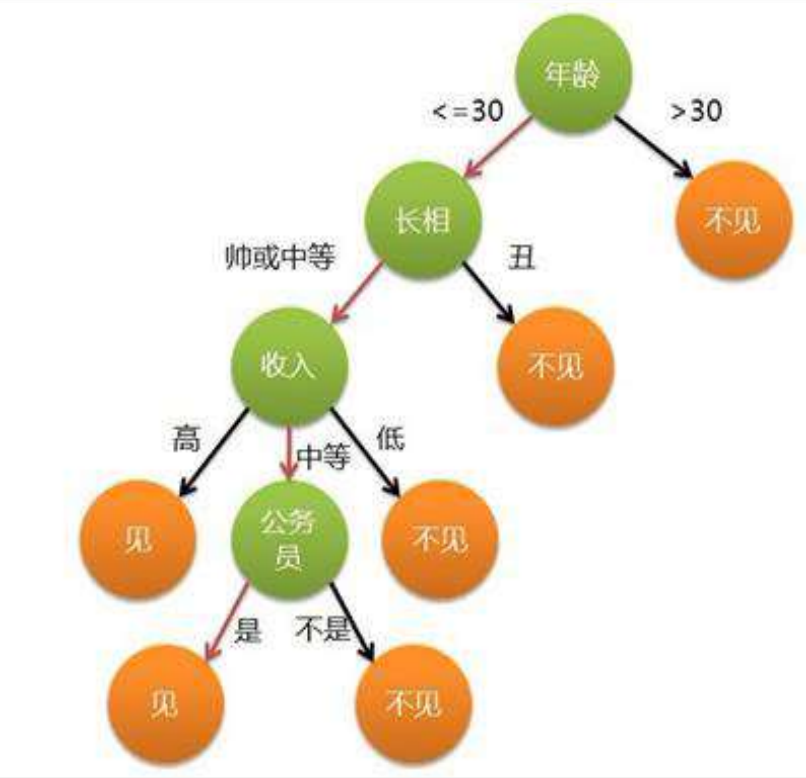
$$= \min_{p(x)} [-H(X) + \log N]$$

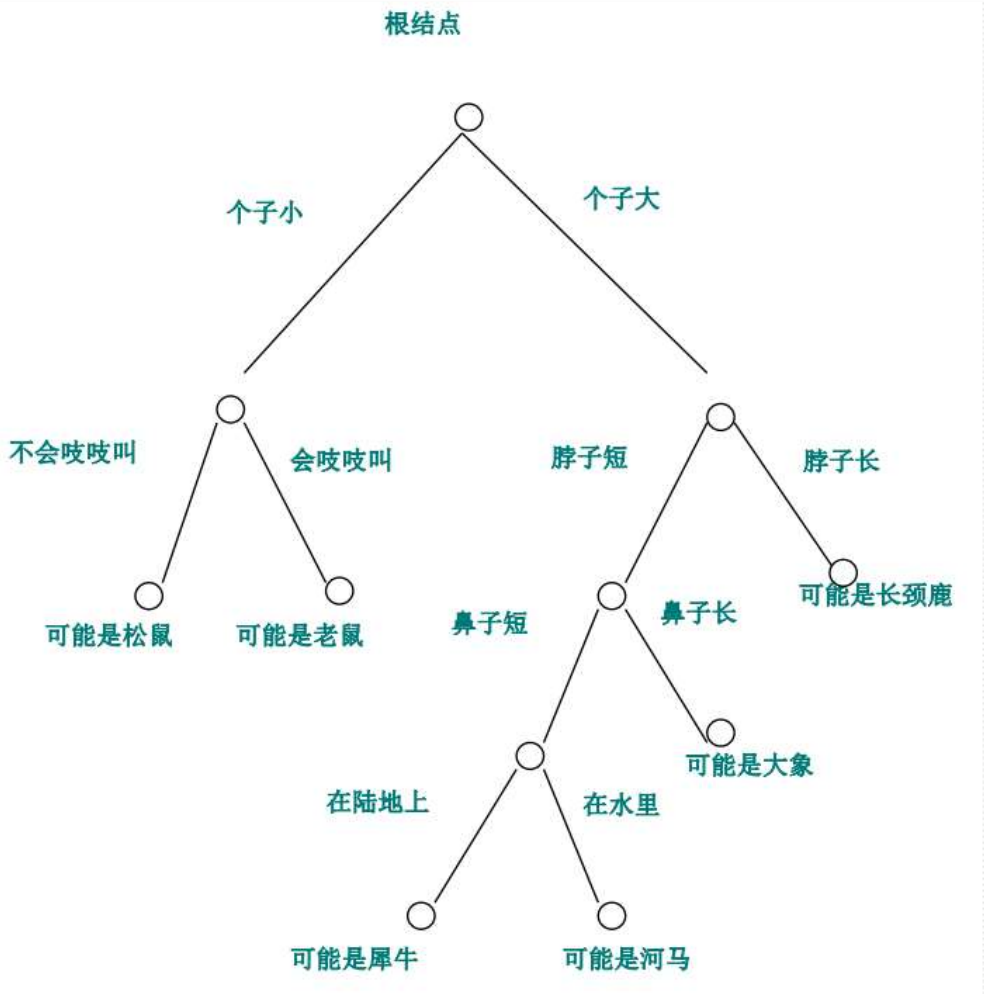
$$= -\left[\max_{p(x)} H(X) - \log N\right]$$

先验概率分布为等概率时，二者具有等价性。

# 决策树基础

决策树示意图



# 决策树基础

决策树(Decision Tree)

◆决策树是一种树型结构，其中每个内部结点表示在一个属性上的测试，每个分支代表一个测试输出，每个叶结点代表一种类别。

◆决策树学习是以实例为基础的归纳学习。

◆决策树学习采用的是自顶向下的递归方法，其基本思想是以信息熵为度量构造一棵熵值下降最快的树，到叶子节点处的熵值为零，此时每个叶节点中的实例都属于同一类。

决策树学习算法的最大优点是，它可以自学习。在学习的过程中，不需要使用者了解过多背景知识，只需要对训练实例进行较好的标注，就能够进行学习。

- 属于有监督学习。
- 从一类无序、无规则的事物(概念)中推理出决策树表示的分类规则。

# 决策树基础

决策树算法主要有以下三种

◆ID3（1986年，Quinlan）

◆C4.5（1993， Quinlan ）

◆CART （1984年，Breiman，Friedman，Olshen，Stone）

- ID3的思想
  - 自顶向下构造决策树
  - 从“哪一个属性将在树的根节点被测试”开始
  - 使用统计测试来确定每一个实例属性单独分类训练样例的能力
- ID3的过程
  - 分类能力最好的属性被选作树的根节点
  - 根节点的每个可能值产生一个分支
  - 训练样例排列到适当的分支
  - 重复上面的过程直到所有训练样本使用完毕

# 决策树基础

- 训练数据批处理，自顶向下递归构造决策树
- DTree(examples, attributes)

  If 所有样本属于同一分类，返回标号为该分类的叶结点
  Else if 属性值为空，返回标号为最普遍分类的叶结点
  Else 选取一个属性，A，作为根结点
    For A的每一个可能的值$v_i$
      令$examples_i$为具有$A=v_i$的样本子集
      从根结点出发增加分支（$A=v_i$）
      如果$examples_i$为空
        则创建标号为最普遍分类的叶结点
        否则递归创建子树——调用
  DTree($examples_i$,attributes-{A})

# 决策树基础

- 概念：当熵和条件熵中的概率由数据估计(特别是极大似然估计)得到时，所对应的熵和条件熵分别称为经验熵和经验条件熵。
- 信息增益表示得知特征A的信息而使得类X的信息的不确定性减少的程度。
- 定义：特征A对训练数据集D的信息增益g(D,A)，定义为集合D的经验熵H(D)与特征A给定条件下D的经验条件熵H(D|A)之差，即：
  - ✓ $g(D,A)=H(D)-H(D|A)$
  - ✓ 显然，这即为训练数据集D和特征A的互信息。

# 决策树基础

- 计算数据集D的经验熵 $$H(D) = -\sum_{k=1}^{K} \frac{|C_k|}{|D|} \log \frac{|C_k|}{|D|}$$
- 遍历所有特征，对于特征A：
  - 计算特征A对数据集D的经验条件熵H(D|A)
  - 计算特征A的信息增益：g(D,A)=H(D) – H(D|A)
  - 选择信息增益最大的特征作为当前的分裂特征

# 决策树基础

## 经验条件熵H(D|A)

$$H(D \mid A) = -\sum_{i,k} p(D_k, A_i) \log p(D_k \mid A_i)$$

$$= -\sum_{i,k} p(A_i) p(D_k \mid A_i) \log p(D_k \mid A_i)$$

$$= -\sum_{i=1}^{n} \sum_{k=1}^{K} p(A_i) p(D_k \mid A_i) \log p(D_k \mid A_i)$$

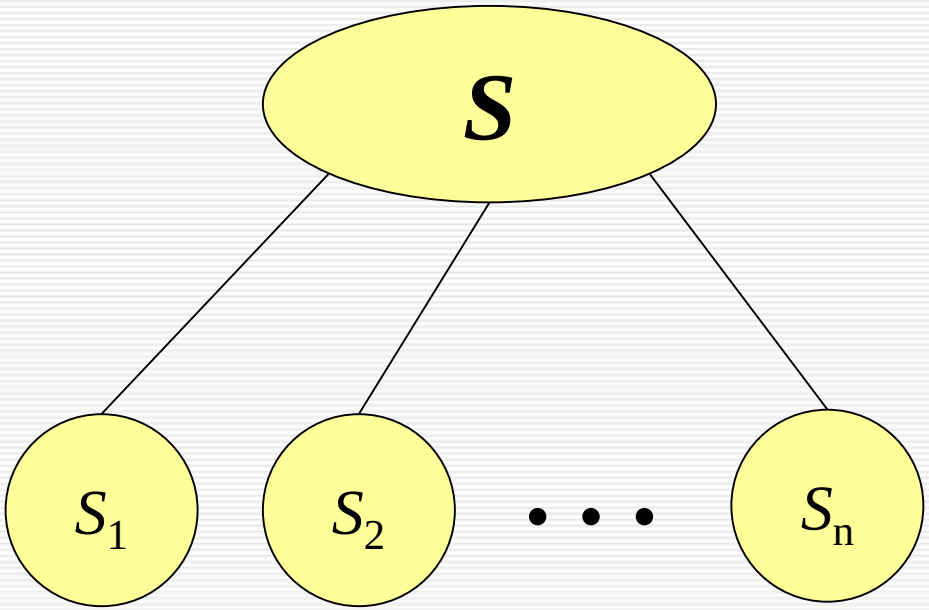
$$= -\sum_{i=1}^{n} p(A_i) \sum_{k=1}^{K} p(D_k \mid A_i) \log p(D_k \mid A_i)$$

$$= -\sum_{i=1}^{n} \frac{|D_i|}{|D|} \sum_{k=1}^{K} \frac{|D_{ik}|}{|D_i|} \log \frac{|D_{ik}|}{|D_i|}$$

# 决策树基础

$$Gain(S,A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

$$Attribute \quad A = \{v_1, \ldots, v_n\}$$



# 决策树基础

## 2. 例子2：判断东西方人

- 属性：身高、眼睛、头发

(tall, blond, blue) w
(short, silver, blue) w
(short, black, blue) w
(tall, blond, brown) w
(tall, silver, blue) w
(short, blond, blue) w

(short, black, brown) e
(tall, silver, black) e
(short, black, brown) e
(tall, black, brown) e
(tall, black, black) e
(short, blond, black) e

# 决策树基础

## 例子2

- 看看身高的信息增益

$$Gain(S,A) = Entropy(S) - \sum_{v \in A} \frac{|S_v|}{|S|} Entropy(S_v)$$

**Q/A**

| | |
|---|---|
| (tall, blond, blue) w | (short, black, brown) e |
| (short, silver, blue) w | (tall, silver, black) e |
| (short, black, blue) w | (short, black, brown) e |
| (tall, blond, brown) w | (tall, black, brown) e |
| (tall, silver, blue) w | (tall, black, black) e |
| (short, blond, blue) w | (short, blond, black) e |

$$Entropy(S) = 1$$

*tall*

| | |
|---|---|
| (tall, blond, blue) w | (tall, silver, black) e |
| (tall, blond, brown) w | (tall, black, brown) e |
| (tall, silver, blue) w | (tall, black, black) e |

$$Entropy(S_{tall}) = 1$$

*short*

| | |
|---|---|
| (short, silver, blue) w | (short, black, brown) e |
| (short, black, blue) w | (short, black, brown) e |
| (short, blond, blue) w | (short, blond, black) e |

$$Entropy(S_{short}) = 1$$

$$Gain(S, Height) = 0$$

# 决策树基础

## 例子2

- 头发？

Q/A

$$Gain(S,A) = Entropy(S) - \sum_{v \in A} \frac{|S_v|}{|S|} Entropy(S_v)$$

(tall, blond, blue) w
(short, silver, blue) w
(short, black, blue) w
(tall, blond, brown) w
(tall, silver, blue) w
(short, blond, blue) w
(short, black, brown) e
(tall, silver, black) e
(short, black, brown) e
(tall, black, brown) e
(tall, black, black) e
(short, blond, black) e

$Entropy(S) = 1$

*blond* *silver* *black*

(tall, blond, blue) w
(tall, blond, brown) w
(short, blond, blue) w
(short, blond, black) e

$Entropy(S_{blond}) = 0.811278$

(short, silver, blue) w
(tall, silver, blue) w
(tall, silver, black) e

$Entropy(S_{silver}) = 0.918296$

(short, black, blue) w
(short, black, brown) e
(short, black, brown) e
(tall, black, brown) e
(tall, black, black) e

$Entropy(S_{black}) = 0.721928$

$$Gain(S, Hair) = 0.199197$$

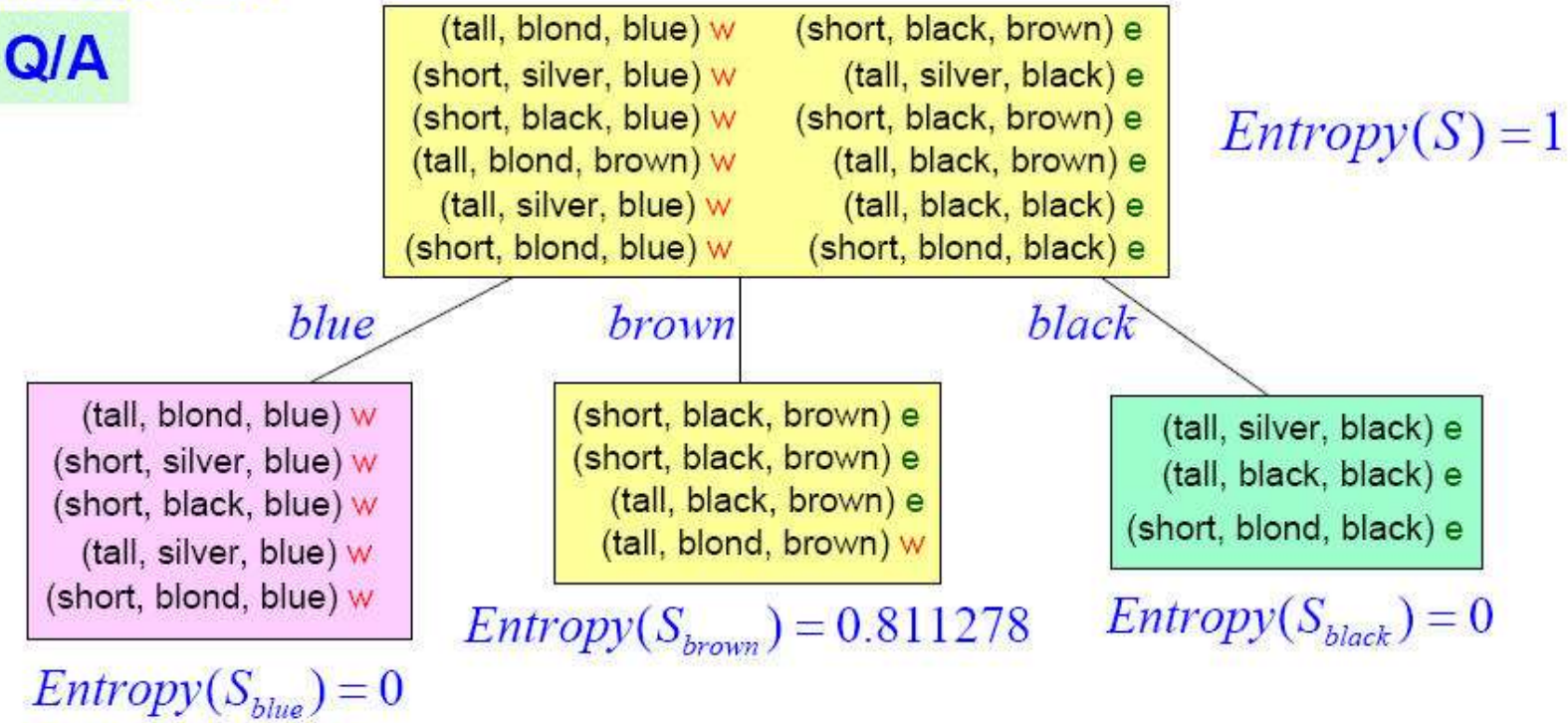
# 决策树基础

## 例子2

$$Gain(S,A) = Entropy(S) - \sum_{v \in A} \frac{|S_v|}{|S|} Entropy(S_v)$$

- 眼睛颜色？

**Q/A**

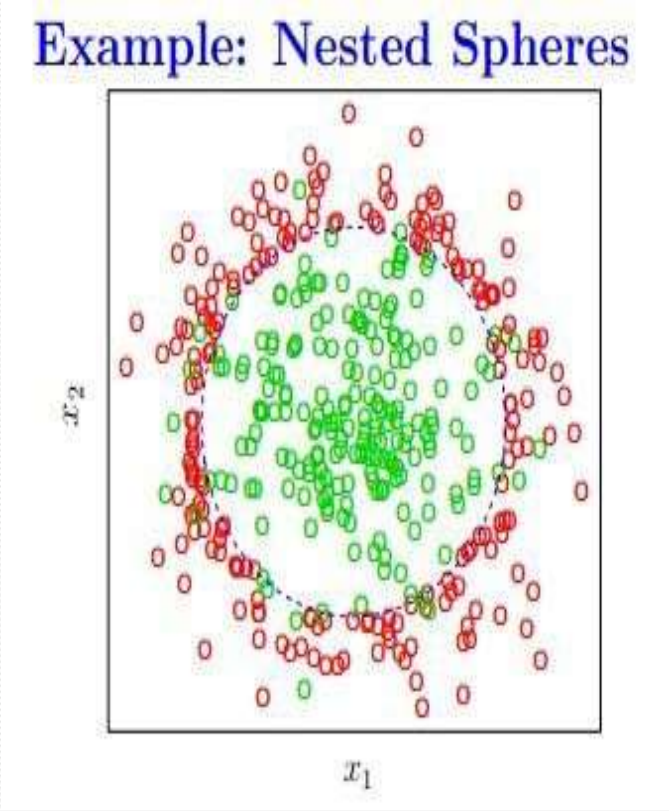



$$Gain(S, Eye) = 0.829574$$

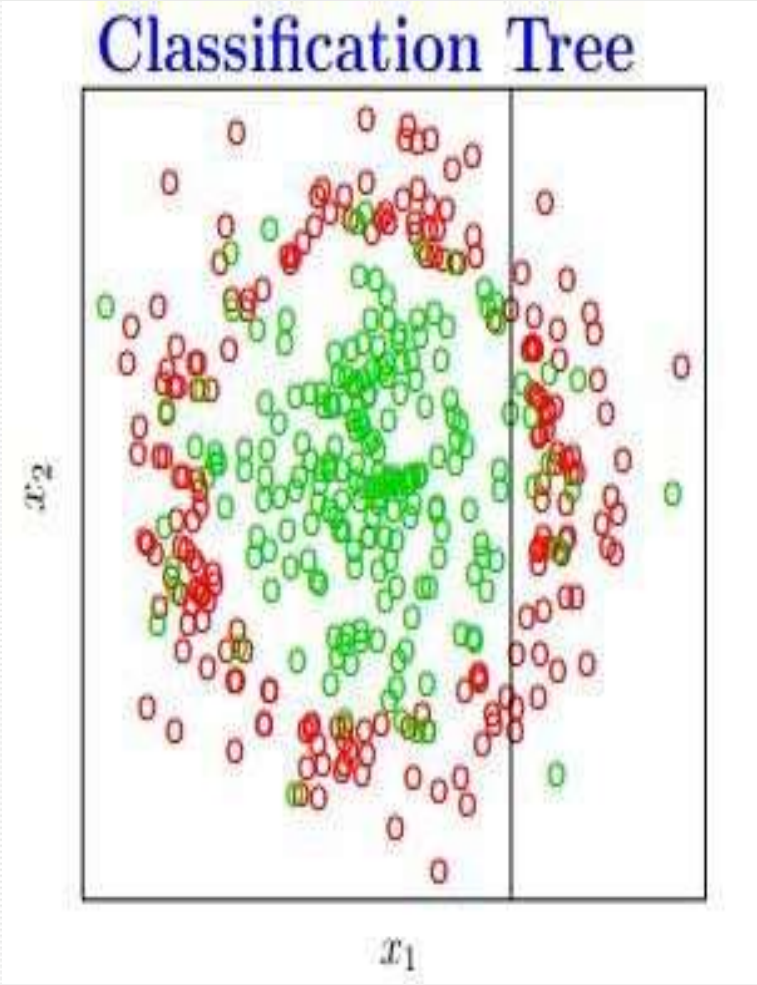
- **Yes! 眼睛颜色!**

# 决策树的生成过程

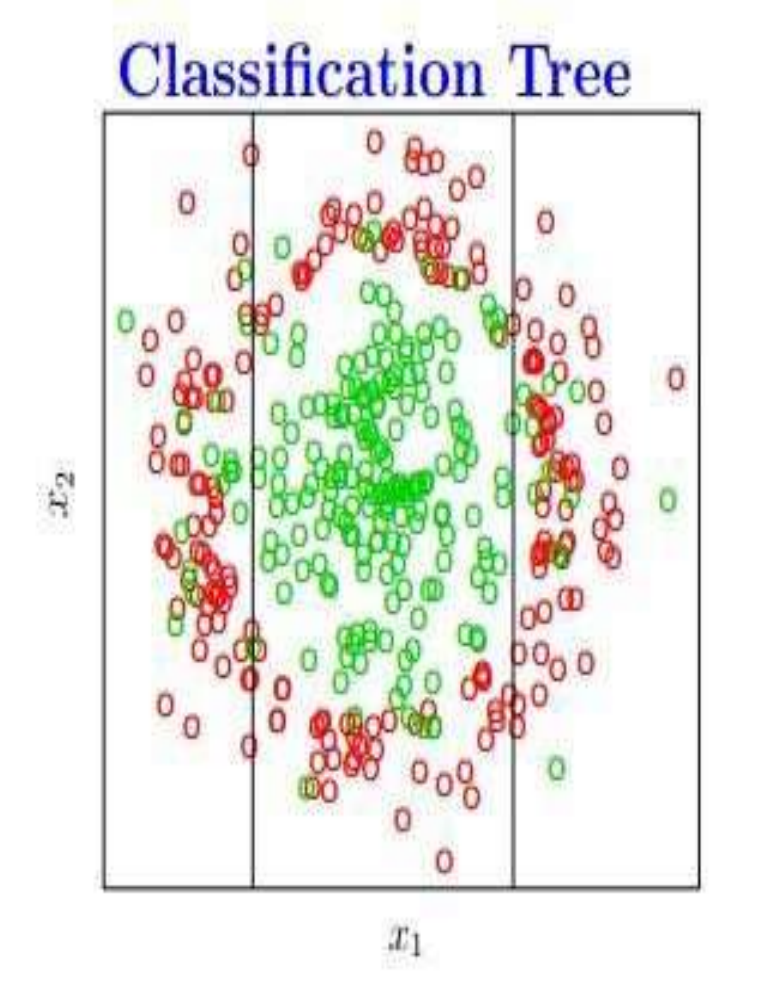
- 对于下面的数据，希望分割成红色和绿色两个类



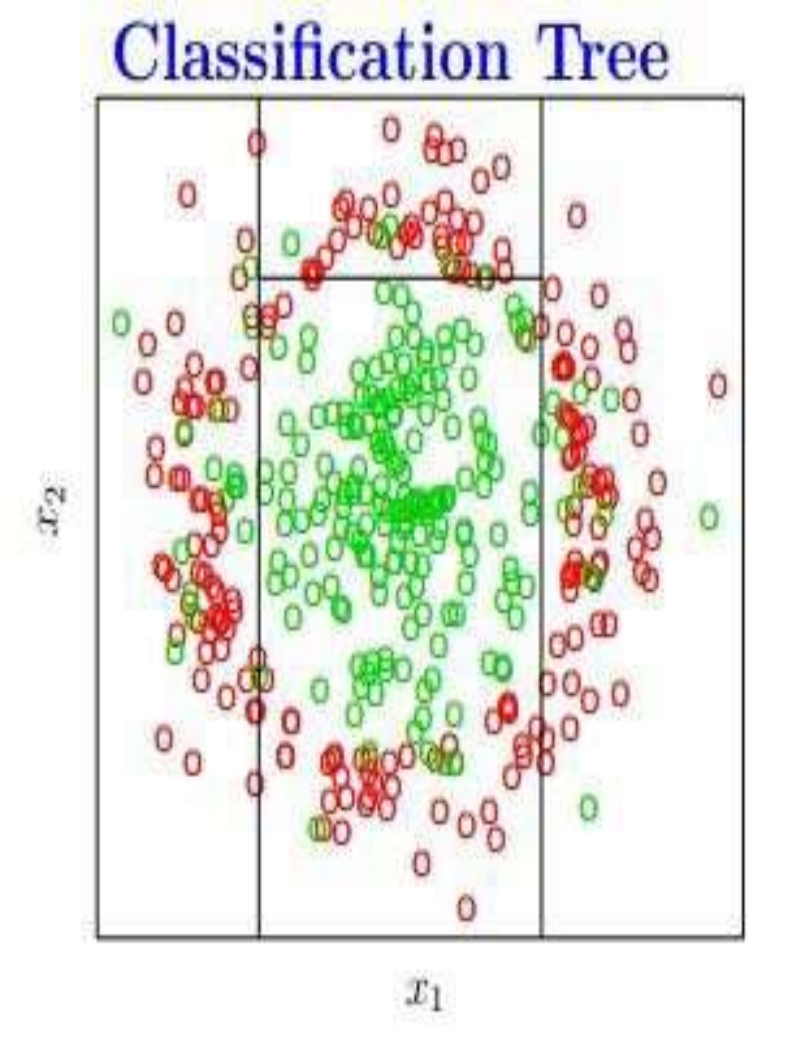
# 决策树的生成过程



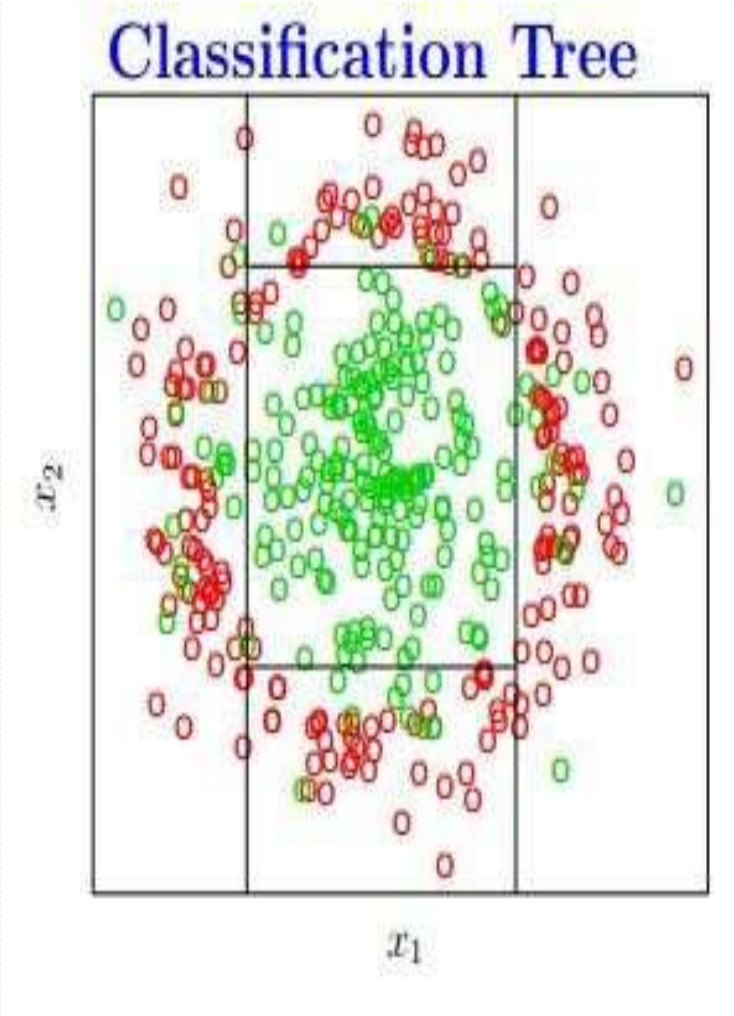
# 决策树的生成过程



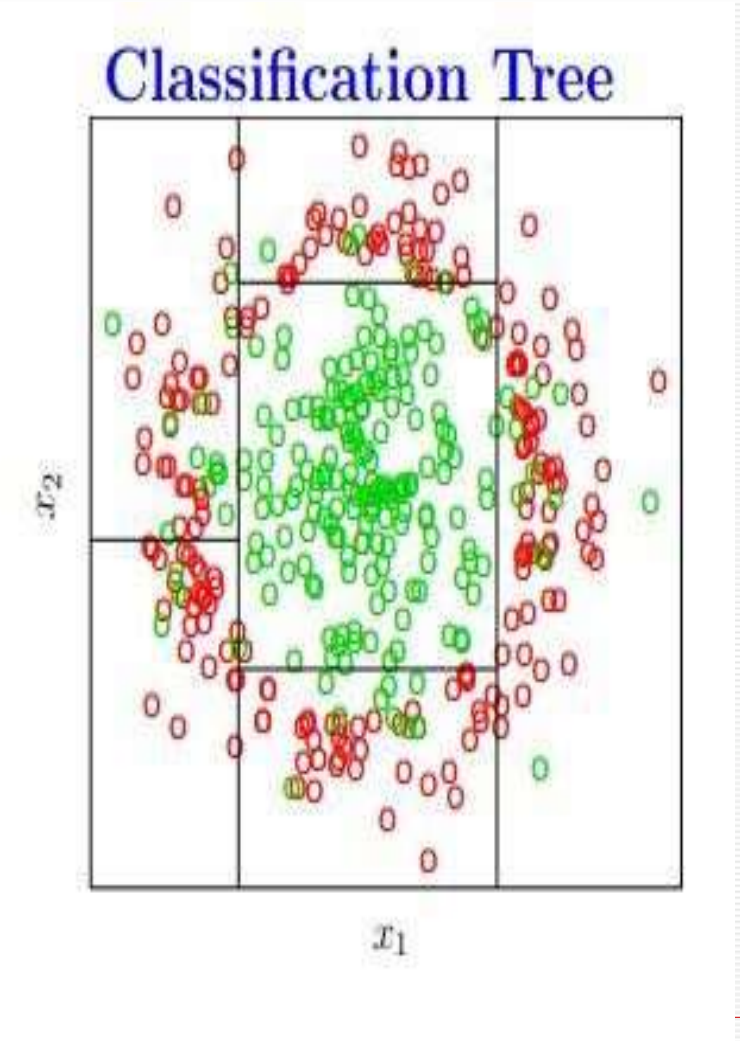
# 决策树的生成过程



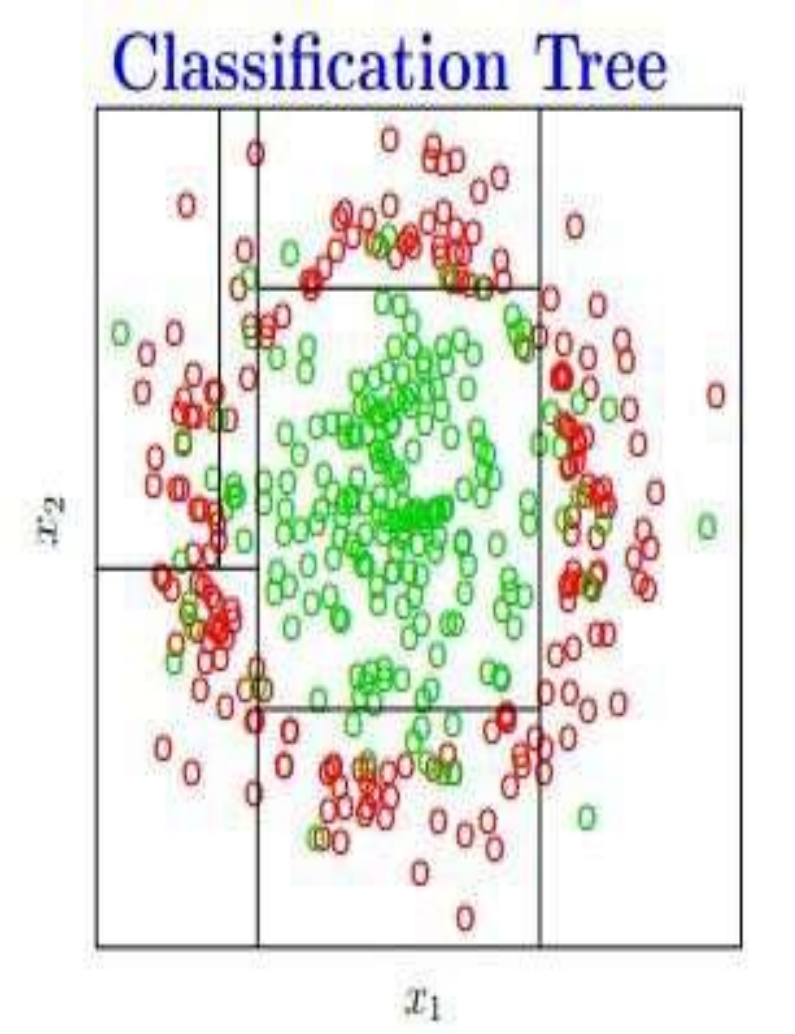
# 决策树的生成过程



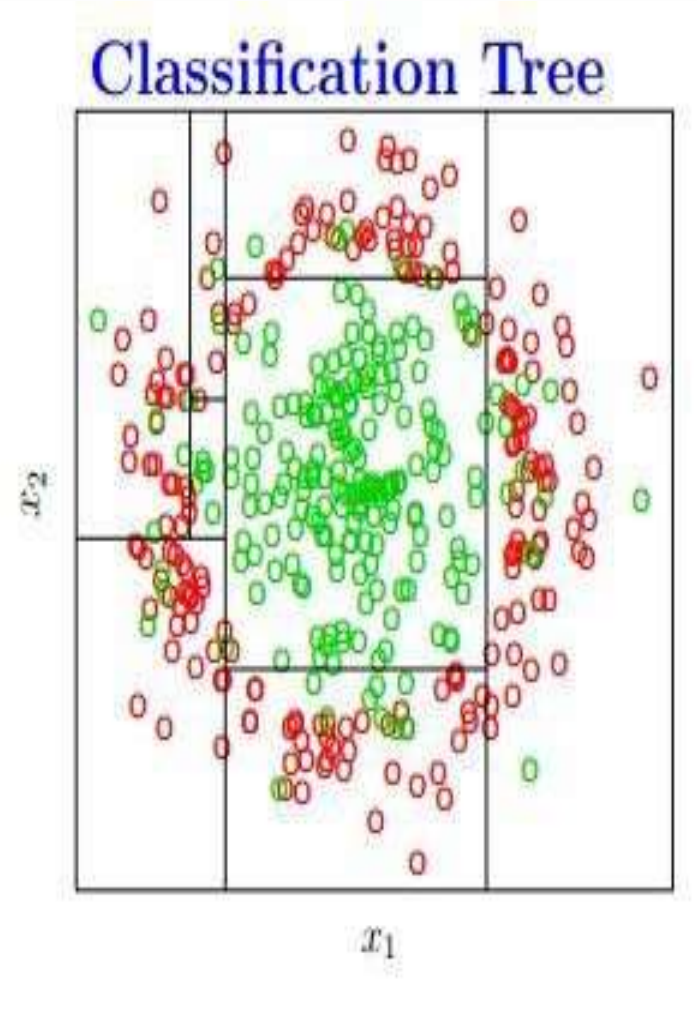
# 决策树的生成过程



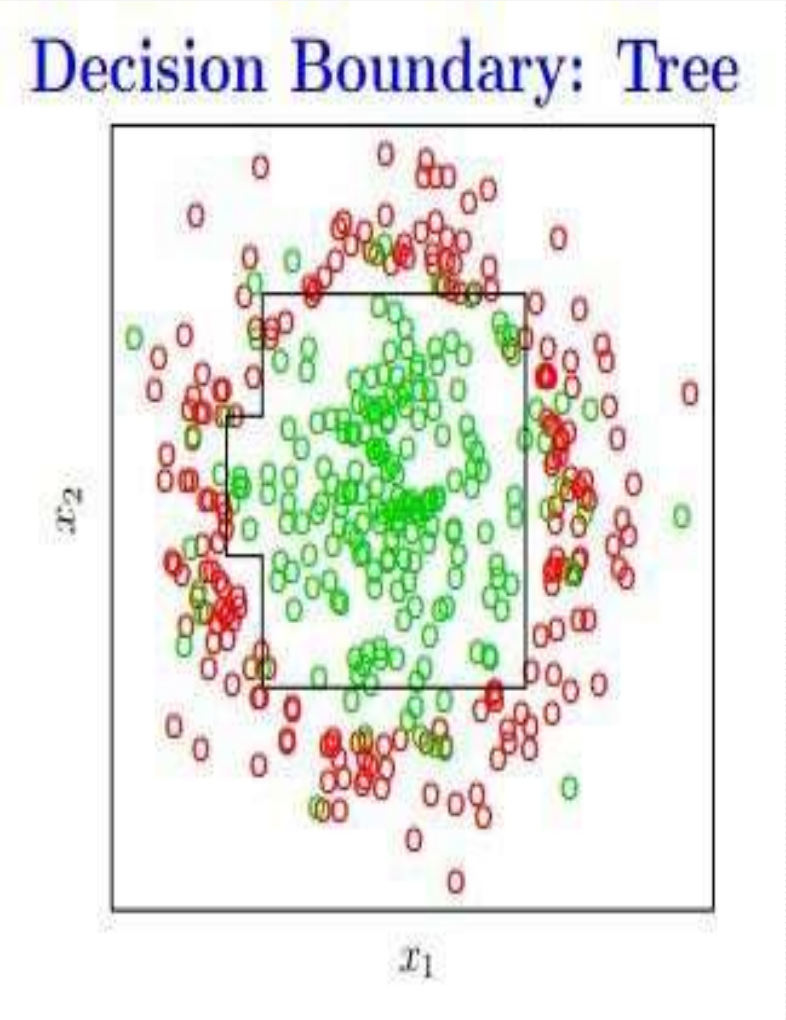
# 决策树的生成过程



# 决策树的生成过程



# 决策树的生成过程



# ID3算法存在的主要问题

- ID3的主要不足
  - 过度拟合问题
  - 处理连续属性值问题
  - 处理缺少属性值问题
  - 属性选择的度量标准问题
  - 处理不同代价的属性值问题
- 针对上述问题，ID3被扩展成C4.5

# 信息增益比率

- 信息增益缺点：偏爱那些有大量值的属性，产生很多小而纯的子集，如病人ID、姓名、日期等
- 要降低这些情况下的增益
- 首先计算与分类无关属性的信息量，即该属性的熵

$$SplitInfo(S, A) = \sum_{i=1}^{c} \frac{|S_i|}{|S|} \log_2 \left( \frac{|S_i|}{|S|} \right)$$

- 其中$S_i$为S中具有属性A第i个值的子集。某属性按值分割样本越平均，则SplitInfo越大
- 增益比率利用SplitInfo来避免选择这些属性

$$GainRatio(S, A) = \frac{Gain(S, A)}{SplitInfo(S, A)}$$

# 信息增益比率

- 然而，当$|S_i|=|S|$时SplitInfo可能很小甚至为0，从而导致信息比率过大甚至溢出
- C4.5对此进行了改进，它计算每个特征的信息增益，对于超过平均增益的特征，再进一步根据增益比率来选取特征

# 信息增益比率

因为特征X被固定以后，给系统带来的增益(或者说为系统减小的不确定度)为：

$$IG(X) = H(c) - H(c|X)$$
$$= -\sum_{i=1}^{n} p(c_i)\log_2 p(c_i) + \sum_{i=1}^{n} p(x = x_i)H(c|x = x_i)$$

特征$X$的熵：

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

特征$X$的信息增益：

$$IG(X) = H(c) - H(c|X)$$

那么信息增益比为：

$$g_r = \frac{H(c) - H(c|X)}{H(X)}$$

在决策树算法中，ID3使用信息增益，c4.5使用信息增益比。

# Gini系数

Gini系数是一种与信息熵类似的做特征选择的方式，可以用来数据的不纯度。在CART(Classification and Regression Tree)算法中利用基尼指数构造二叉决策树。

Gini系数的计算方式如下：

$$Gini(D) = 1 - \sum_{i=1}^{n} p_i^2$$

其中，D表示数据集全体样本，$p_i$表示每种类别出现的概率。取个极端情况，如果数据集中所有的样本都为同一类，那么有$p_0 = 1$，$Gini(D) = 0$，显然此时数据的不纯度最低。

与信息增益类似，我们可以计算如下表达式：

$$\Delta Gini(X) = Gini(D) - Gini_X(D)$$

# 决策树的过拟合问题

决策树对训练属于有很好的分类能力，但对未知的测试数据未必有好的分类能力，泛化能力弱，即可能发生过拟合现象。

- 剪枝
- 随机森林
- XGBoost

# 剪枝

- 三种决策树的剪枝过程算法相同，区别仅是对于当前树的评价标准不同。
  - 信息增益、信息增益率、基尼系数
- 剪枝总体思路：
  - 由完全树$T_0$开始，剪枝部分结点得到$T_1$，再次剪枝部分结点得到$T_2$...直到仅剩树根的树$T_k$；
  - 在验证数据集上对这k个树分别评价，选择损失函数最小的树$T_\alpha$

# 剪枝系数的确定

- 根据原损失函数 $C(T)=\sum_{t\in leaf} N_t \cdot H(t)$
- 叶结点越多，决策树越复杂，损失越大，修正：
  - 当α=0时，未剪枝的决策树损失最小； $C_\alpha(T)=C(T)+\alpha\cdot|T_{leaf}|$
  - 当α=+∞时，单根结点的决策树损失最小。
- 假定当前对以r为根的子树剪枝：
  - 剪枝后，只保留r本身而删掉所有的叶子
- 考察以r为根的子树：
  - 剪枝后的损失函数：$C_\alpha(r)=C(r)+\alpha$
  - 剪枝前的损失函数：$C_\alpha(R)=C(R)+\alpha\cdot|R_{leaf}|$
  - 令二者相等，求得： $\alpha=\frac{C(r)-C(R)}{<R_{leaf}>-1}$
- α称为结点r的剪枝系数。

$T$表示树

$C(T)$表示当前损失

$T_{leaf}$表示分裂后的叶子节点数

$\alpha \geq 0$为参数

$C_\alpha(T)$表示损失函数的正则化

$\alpha|T_{leaf}|$实际上是损失函数的正则项

$$C(T)=\sum_{t=1}^{|T_{leaf}|} N_t H_t(T)$$

$N_t$表示该叶结点含有的样本点个数

那么，$C_\alpha(T)=\sum_{t=1}^{|T_{leaf}|} N_t H_t(T)+\alpha|T_{leaf}|$

# 剪枝算法

- 对于给定的决策树$T_0$：
  - 计算所有内部节点的剪枝系数；
  - 查找最小剪枝系数的结点，剪枝得决策树$T_k$ ；
  - 重复以上步骤，直到决策树$T_k$只有1个结点；
  - 得到决策树序列$T_0T_1T_2...T_k$ ；
  - 使用验证样本集选择最优子树。
- 使用验证集做最优子树的标准，可以使用评价函数：

$$C(T) = \sum_{t \in leaf} N_t \cdot H(t)$$

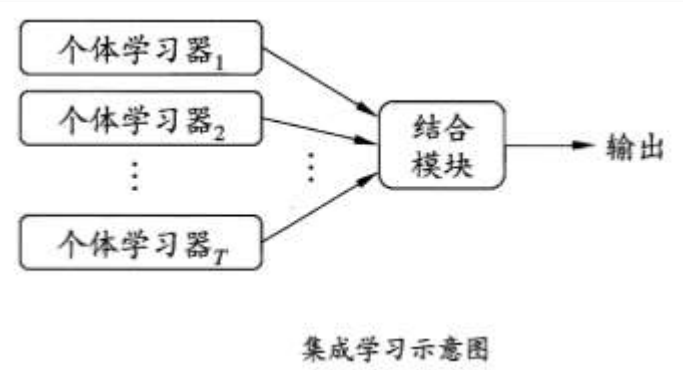
# 剪枝算法

- 需要修剪时的两个基本方法
  - 先剪枝：通过提前停止树的构造——如果在一个节点划分样本将导致低于预定义临界值的分裂（e.g. 使用信息增益度量）
    - 选择一个合适的临界值往往很困难
  - 后剪枝：由“完全生长”的树剪去分枝——对于树中的每个非树叶节点，计算该节点上的子树被剪枝可能出现的期望错误率
    - 使用一个独立的测试集来评估每颗树的准确率，就能得到具有最小期望错误率的决策树

# 集成学习

个体学习器通常是用一个现有的学习算法从训练数据产生，例如C4.5决策树算法、BP神经网络算法等。此时集成中只包含同种类型的个体学习器，例如“决策树集成”中的个体学习器全是决策树，“神经网络集成”中就全是神经网络，这样的集成是“同质”（homogeneous）的，同质集成中的个体学习器也称为“基学习器”（base learner），相应的学习算法称为“基学习算法”（base learning algorithm）。

异质（heterogeneous）集成算法，就是包含不同类型的个体学习器，例如同时包含决策树和神经网络，那么这时个体学习器一般不称为基学习器，而称作“组件学习器”（component leaner）或直接称为个体学习器。



集成学习示意图

# 集成学习

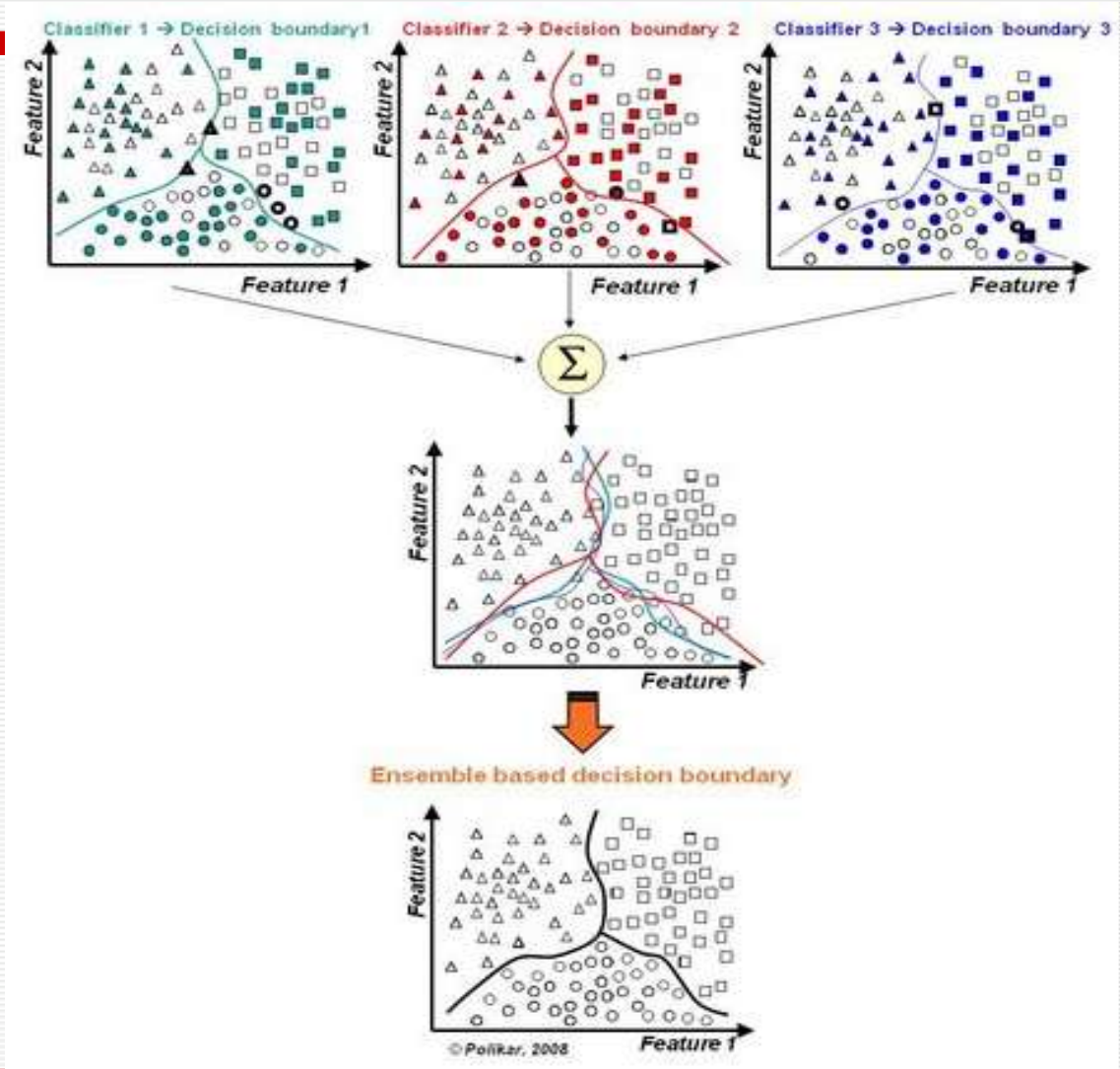



Figure 1: Combining an ensemble of classifiers for reducing classification error and/or model selection.

# 集成学习

假设有一个二分类问题，其中$y \in -1, +1$，以及真实函数f，假定及分类器的错误率为$\epsilon$，即对于每个基分类器$h_i$有：

$$P(h_i(x) \neq f(x)) = \epsilon。$$

$$P(|\bar{X} - E(\bar{X})| \geq \delta) \leq \exp(-2\delta^2 n^2)$$

假设基分类器的错误率相互独立，则由Hoeffding不等式可知，集成的错误率为：

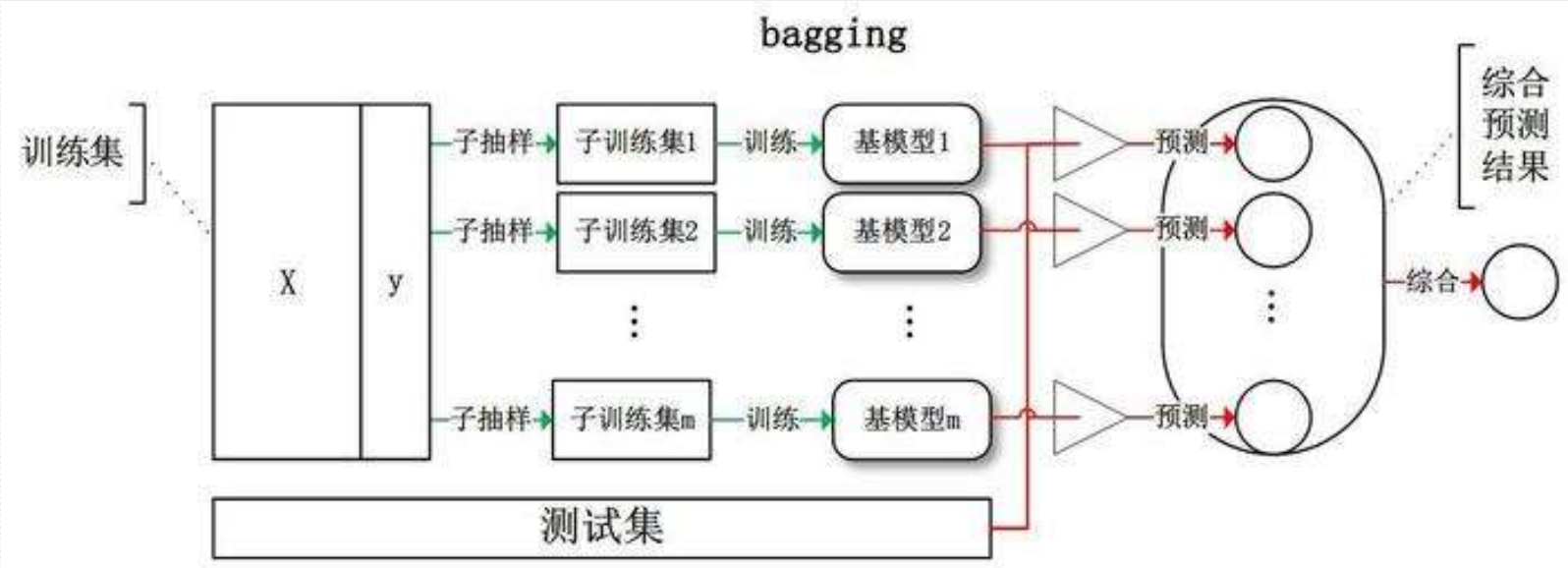
$$P(H(x) \neq f(x)) = \Sigma_{k=1}^{\lceil T/2 \rceil} C_T^k (1-\epsilon)^k \epsilon^{T-k} \leq exp(-\frac{1}{2}T(1-2\epsilon)^2)。$$

上式表明，随着集成个体分类器数目T的增大，集成的错误率将指数级下降，最终趋向于0。

上述式子的推导是基于一个关键假设：基学习器的误差相互独立。然而现实情况是，个体学习器都是为解决同一个问题训练出来的，它们之间显然不可能相互独立。而事实上，个体学习器的"准确性"和"多样性"本身就存在冲突。一般的，准确性很高之后，要增加多样性就需牺牲准确性。而如何产生并结合"好而不同"的个体学习器，恰是集成学习研究的核心。

# 集成学习-Bagging

Bagging基于前面提到过的自助采样法（bootstrap sampling）。给定包含m个样本的数据集，先随机取出一个样本放入采样集中，再把该样本放回初始数据集，使得下次采样时该样本仍有可能被选中，这样，经过m此随机采样操作，得到含m个样本的采样集，初始训练集中有的样本在采样集里多次出现，有的则从未出现。



# 集成学习-Bagging

Bootstrap方法：它是基于统计学的放回抽样数据集的方法来形成训练集。

一个有n个实例的数据集进行了n次的放回抽样，从而得到另一个拥有n个实例的训练数据集，新生成的数据集中会有一些重复的实例，那么在原始数据集中必然有部分实例未被抽样，这些未被抽样的实例作为测试实例。

- 训练数据集和测试数据集的处理
  - 自引导法中假设进行n次抽取，某个实例不被抽样到训练集中的概率为：
  - $\lim_{n\to\infty}\left(1-\frac{1}{n}\right)^n = e^{-1} \approx 0.368$
  - 分类误差为：
  - $Error = 0.632 * Error_{\text{训练样例}} + 0.368 * Error_{\text{测试样例}}$

# 集成学习-Bagging

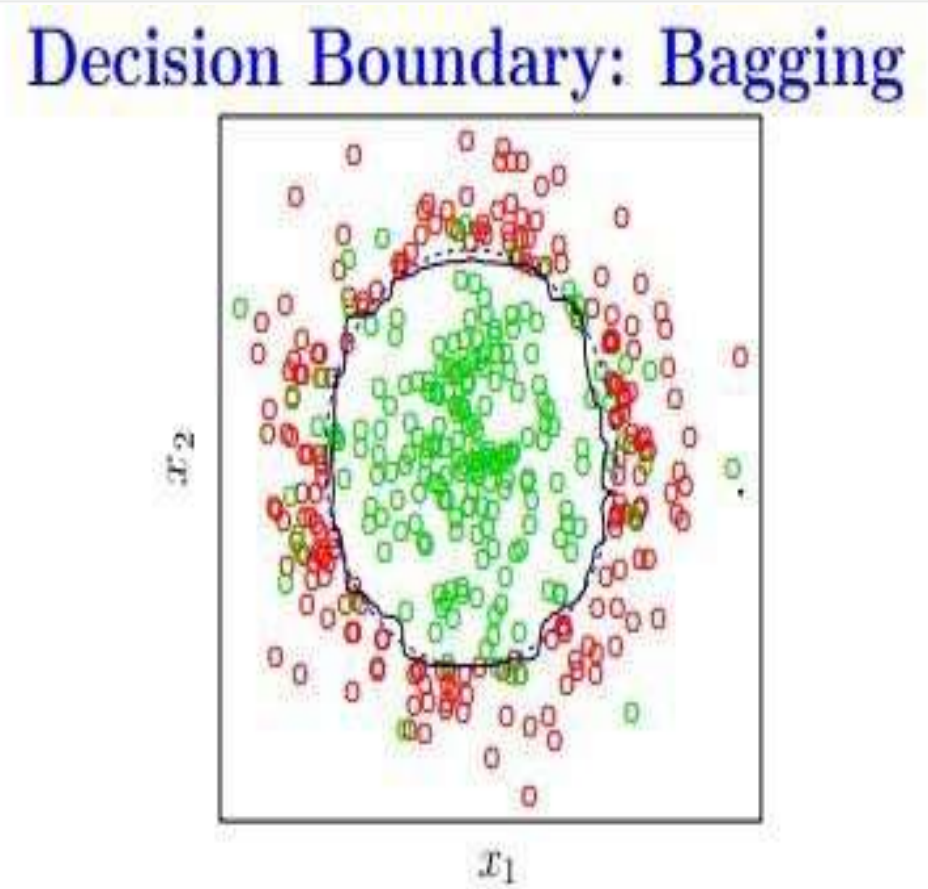
- 随机森林在bagging基础上做了修改：
  - 从样本集中用Bootstrap采样选出n个样本；
  - 从所有属性中随机选择k个属性，选择最佳分割属性作为节点建立决策树；
  - 重复以上两步m次，即建立了m棵决策树，每棵决策树都最大可能地进行生长而不进行剪枝，许多决策树构成一片森林，决策树之间没有联系。
  - 这m个形成随机森林，通过投票表决结果，决定数据属于哪一类。

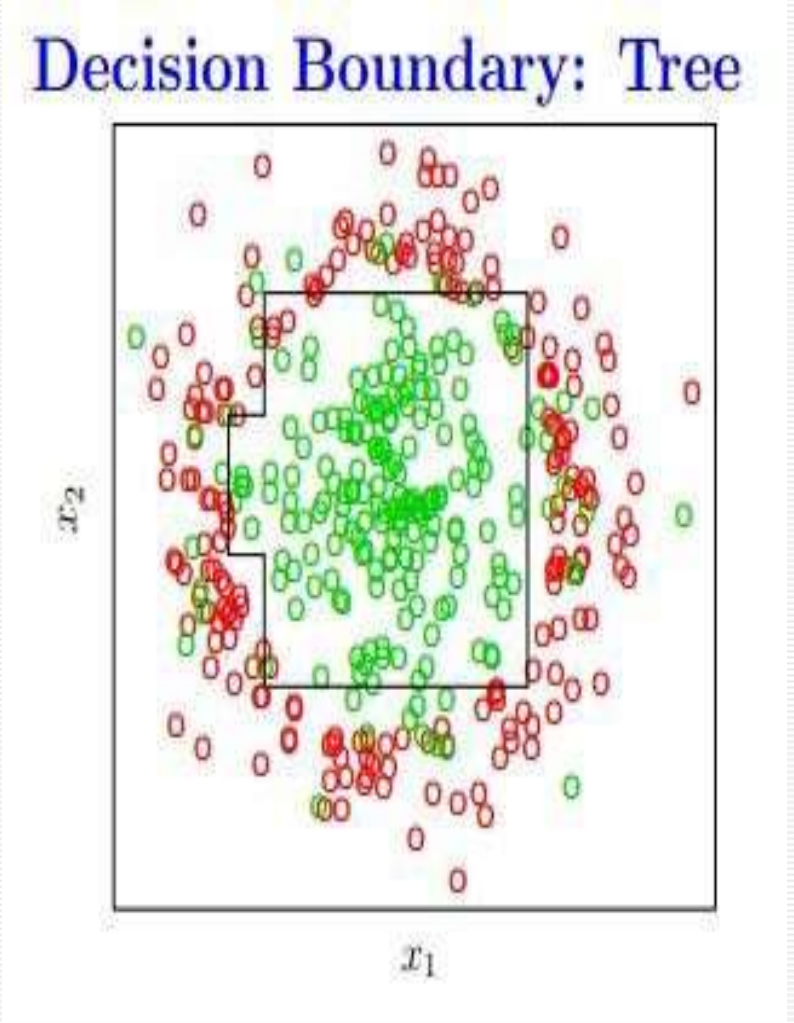
# 集成学习-Bagging

- 优点：
- 不容易出现过拟合，因为选择训练样本的时候就不是全部样本。
- 可以既可以处理属性为离散值的量，比如ID3算法来构造树，也可以处理属性为连续值的量，比如C4.5算法来构造树。
- 对于高维数据集的处理能力令人兴奋，它可以处理成千上万的输入变量，并确定最重要的变量，因此被认为是一个不错的降维方法。此外，该模型能够输出变量的重要性程度，这是一个非常便利的功能。
- 分类不平衡的情况时，随机森林能够提供平衡数据集误差的有效方法。

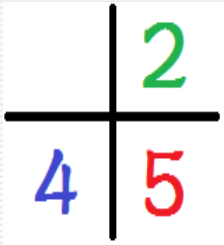
- 缺点：
- 当进行回归时，随机森林不能够作出超越训练集数据范围的预测，这可能导致在对某些有特定噪声的数据进行建模时出现过度拟合。
- 对数据集的数据规模要求比普通决策树要求高。
- 对于许多统计建模者来说，随机森林给人的感觉像是一个黑盒子——无法控制模型内部的运行，只能在不同的参数和随机种子之间进行尝试。

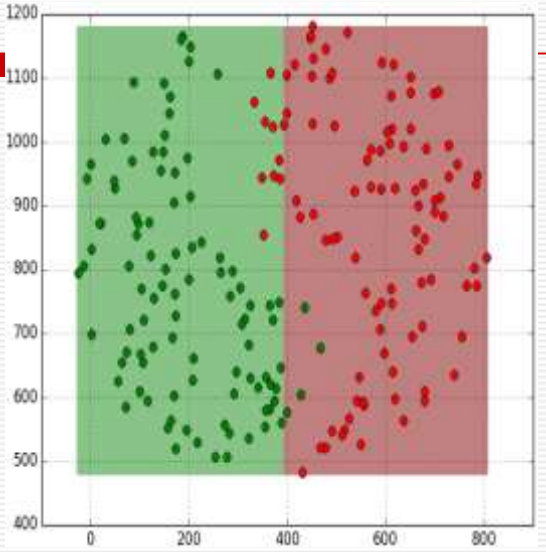
# Bagging的结果

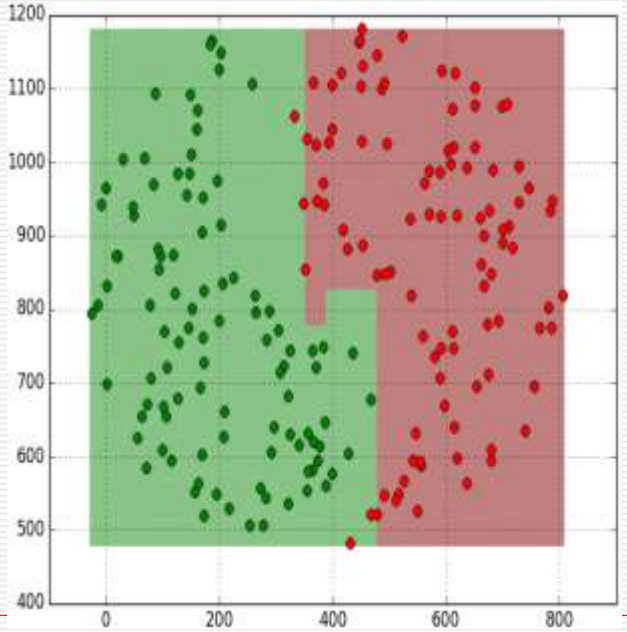

Decision Boundary: Bagging
$x_2$
$x_1$

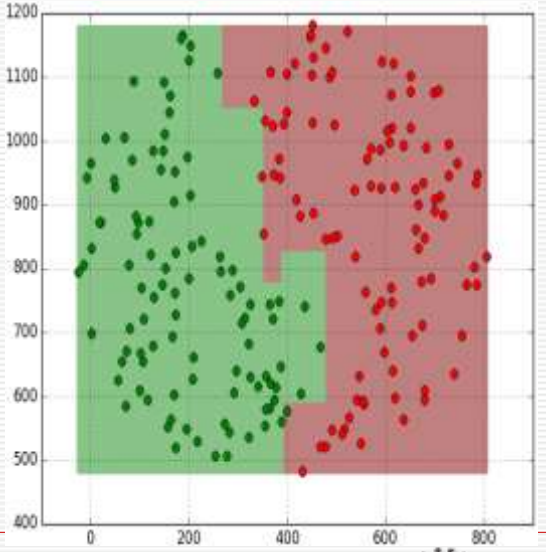

Decision Boundary: Tree
$x_2$
$x_1$

# 决策树：Level

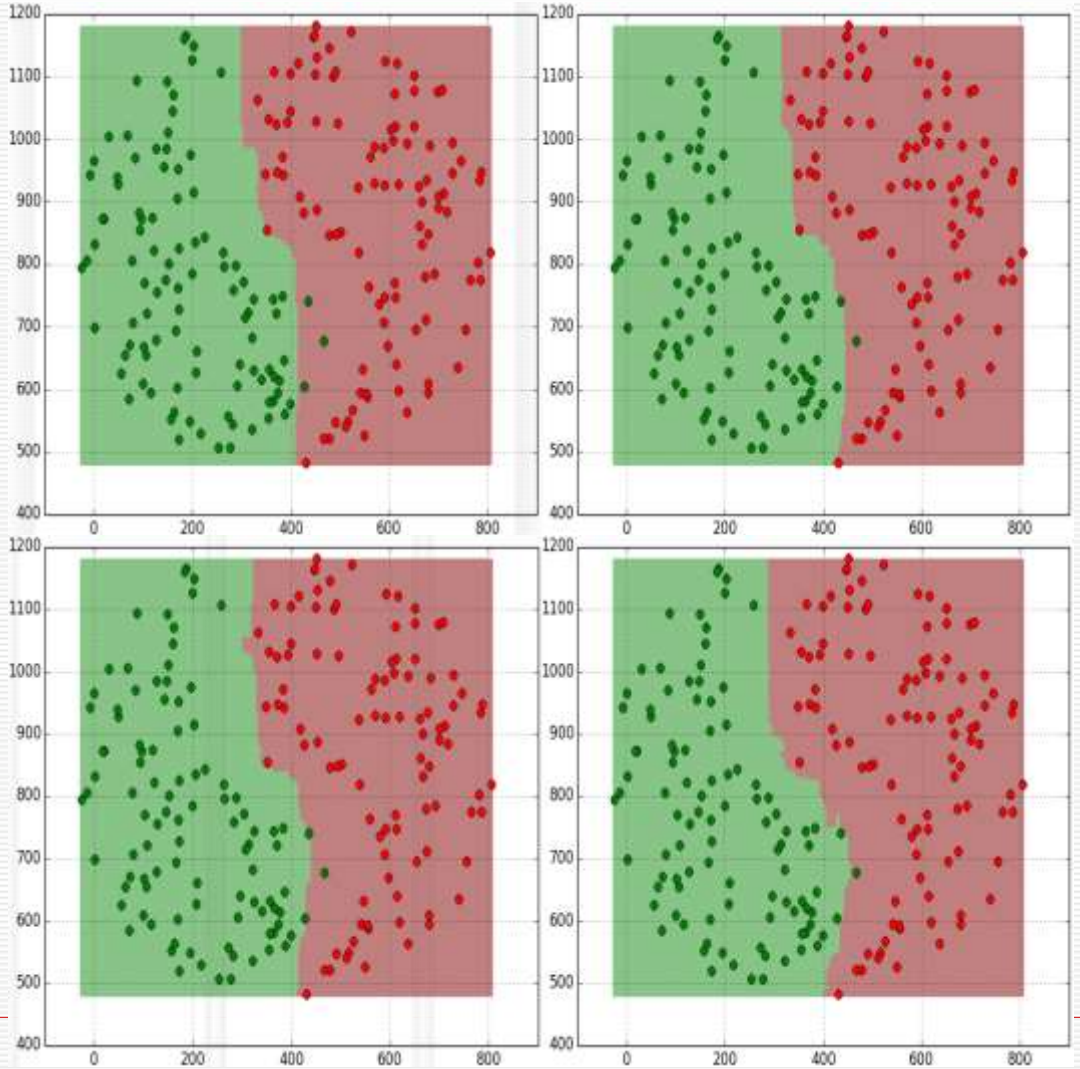

2
4
5

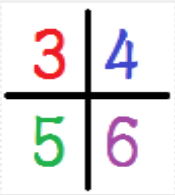

1200
1100
1000
900
800
700
600
500
400
0
200
400
600
800


1200
1100
1000
900
800
700
600
500
400
0
200
400
600
800

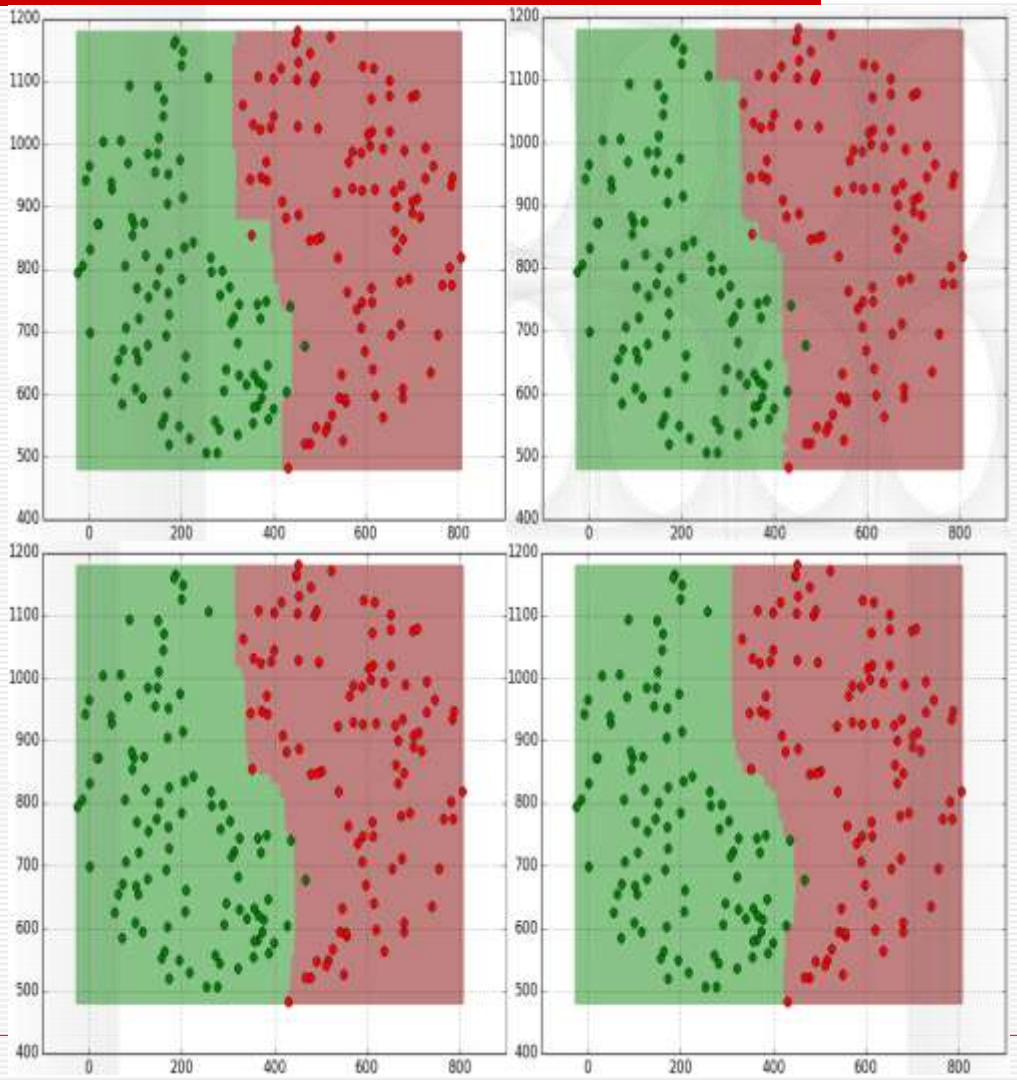

1200
1100
1000
900
800
700
600
500
400
0
200
400
600
800

# 随机森林：30，Level



# 随机森林：4，Tree



| | |
|---|---|
| 10 | 20 |
| 30 | 40 |

# 集成学习-Boosting

Boosting族算法最著名的代表是AdaBoost，它的算法描述如下图，其中$y_i \in -1, +1$,f是真实函数。

输入：训练集 $D = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_m, y_m)\}$;
基学习算法 $\mathfrak{L}$;
训练轮数 $T$.

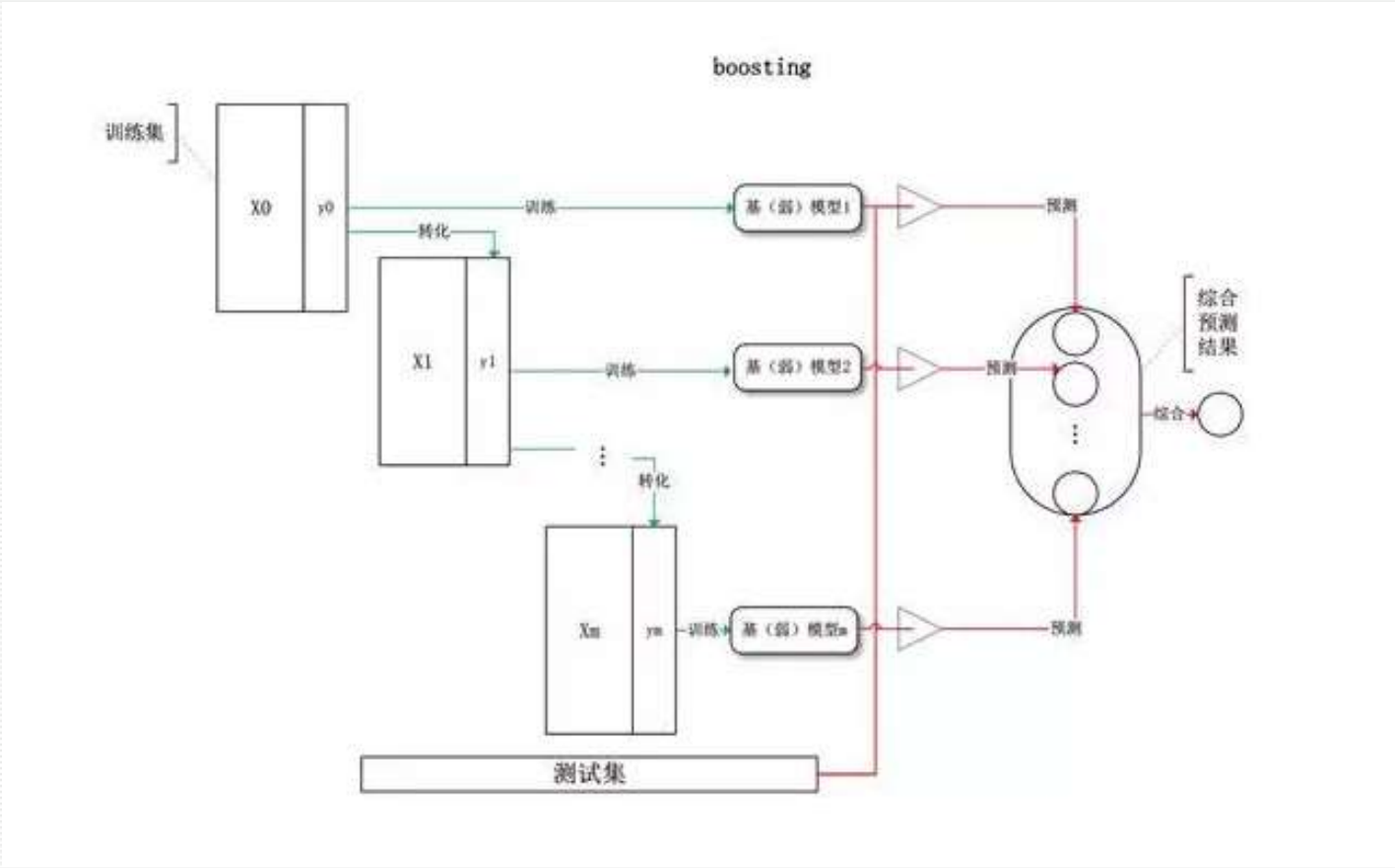
过程：

1: $\mathcal{D}_1(\boldsymbol{x}) = 1/m$. 初始化样本权值分布.
2: **for** $t = 1, 2, \ldots, T$ **do**
3: $h_t = \mathfrak{L}(D, \mathcal{D}_t)$; 基于分布 $\mathcal{D}_t$ 从数据集 $D$ 中训练出分类器 $h_t$.
4: $\epsilon_t = P_{\boldsymbol{x} \sim \mathcal{D}_t}(h_t(\boldsymbol{x}) \neq f(\boldsymbol{x}))$; 估计 $h_t$ 的误差.
5: **if** $\epsilon_t > 0.5$ **then break**
6: $\alpha_t = \frac{1}{2} \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$; 确定分类器 $h_t$ 的权重.
7: $\mathcal{D}_{t+1}(\boldsymbol{x}) = \frac{\mathcal{D}_t(\boldsymbol{x})}{Z_t} \times \begin{cases} \exp(-\alpha_t), & \text{if } h_t(\boldsymbol{x}) = f(\boldsymbol{x}) \\ \exp(\alpha_t), & \text{if } h_t(\boldsymbol{x}) \neq f(\boldsymbol{x}) \end{cases}$
$= \frac{\mathcal{D}_t(\boldsymbol{x}) \exp(-\alpha_t f(\boldsymbol{x}) h_t(\boldsymbol{x}))}{Z_t}$ 更新样本分布，其中 $Z_t$ 是规范化因子，以确保 $\mathcal{D}_{t+1}$ 是一个分布.
8: **end for**
输出：$H(\boldsymbol{x}) = \operatorname{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(\boldsymbol{x})\right)$

图 8.3 AdaBoost算法

# 集成学习-Boosting

Boosting族算法最著名的代表是AdaBoost，它的算法描述如下图，其中$y_i \in -1, +1$,f是真实函数。



# 集成学习-Boosting

AdaBoost算法有多种推到方式，比较容易理解的是基于“加性模型”（additive model），即基学习器的线性组合

$$H(x) = \Sigma_{t=1}^{T} \alpha_t h_t(x)$$

来最小化指数损失函数

$$l_{exp}(H|D) = E_{x\sim D}[e^{-f(x)H(x)}]。$$

# 集成学习-Boosting

若 $H(\boldsymbol{x})$ 能令指数损失函数最小化, 则考虑式(8.5)对 $H(\boldsymbol{x})$ 的偏导

$$\frac{\partial \ell_{\exp}(H \mid \mathcal{D})}{\partial H(\boldsymbol{x})} = -e^{-H(\boldsymbol{x})} P(f(\boldsymbol{x}) = 1 \mid \boldsymbol{x}) + e^{H(\boldsymbol{x})} P(f(\boldsymbol{x}) = -1 \mid \boldsymbol{x}) , \quad (8.6)$$

令式(8.6)为零可解得

$$H(\boldsymbol{x}) = \frac{1}{2} \ln \frac{P(f(\boldsymbol{x}) = 1 \mid \boldsymbol{x})}{P(f(\boldsymbol{x}) = -1 \mid \boldsymbol{x})} , \quad (8.7)$$

因此, 有

$$\begin{aligned} \operatorname{sign}(H(\boldsymbol{x})) &= \operatorname{sign}\left(\frac{1}{2} \ln \frac{P(f(\boldsymbol{x}) = 1 \mid \boldsymbol{x})}{P(f(\boldsymbol{x}) = -1 \mid \boldsymbol{x})}\right) \\ &= \begin{cases} 1, & P(f(\boldsymbol{x}) = 1 \mid \boldsymbol{x}) > P(f(\boldsymbol{x}) = -1 \mid \boldsymbol{x}) \\ -1, & P(f(\boldsymbol{x}) = 1 \mid \boldsymbol{x}) < P(f(\boldsymbol{x}) = -1 \mid \boldsymbol{x}) \end{cases} \\ &= \underset{y \in \{-1,1\}}{\arg\max} P(f(\boldsymbol{x}) = y \mid \boldsymbol{x}) , \end{aligned} \quad (8.8)$$

这里忽略了 $P(f(\boldsymbol{x}) = 1 \mid \boldsymbol{x}) = P(f(\boldsymbol{x}) = -1 \mid \boldsymbol{x})$ 的情形.

这意味着 $\operatorname{sign}(H(\boldsymbol{x}))$ 达到了贝叶斯最优错误率. 换言之, 若指数损失函数最小化, 则分类错误率也将最小化; 这说明指数损失函数是分类任务原本 0/1 损失函数的一致的(consistent)替代损失函数.

替代损失函数的"一致性"参见 6.7 节.

# 集成学习-Boosting

在 AdaBoost算法 中, 第一个基分类器 $h_1$ 是通过直接将基学习算法用于初始数据分布而得; 此后迭代地生成 $h_t$ 和 $\alpha_t$, 当基分类器 $h_t$ 基于分布 $\mathcal{D}_t$ 产生后, 该基分类器的权重 $\alpha_t$ 应使得 $\alpha_t h_t$ 最小化指数损失函数

$$\begin{aligned}\ell_{\exp}\left(\alpha_t h_t \mid \mathcal{D}_t\right) &= \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}_t}\left[e^{-f(\boldsymbol{x})\alpha_t h_t(\boldsymbol{x})}\right] \\ &= \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}_t}\left[e^{-\alpha_t}\mathbb{I}\left(f\left(\boldsymbol{x}\right)=h_t\left(\boldsymbol{x}\right)\right)+e^{\alpha_t}\mathbb{I}\left(f\left(\boldsymbol{x}\right)\neq h_t\left(\boldsymbol{x}\right)\right)\right] \\ &= e^{-\alpha_t}P_{\boldsymbol{x}\sim\mathcal{D}_t}\left(f\left(\boldsymbol{x}\right)=h_t\left(\boldsymbol{x}\right)\right)+e^{\alpha_t}P_{\boldsymbol{x}\sim\mathcal{D}_t}\left(f\left(\boldsymbol{x}\right)\neq h_t\left(\boldsymbol{x}\right)\right) \\ &= e^{-\alpha_t}\left(1-\epsilon_t\right)+e^{\alpha_t}\epsilon_t\ , \end{aligned} \tag{8.9}$$

其中 $\epsilon_t = P_{\boldsymbol{x}\sim\mathcal{D}_t}\big(h_t(\boldsymbol{x})\neq f(\boldsymbol{x})\big)$. 考虑指数损失函数的导数

$$\frac{\partial\ell_{\exp}(\alpha_t h_t \mid \mathcal{D}_t)}{\partial\alpha_t} = -e^{-\alpha_t}(1-\epsilon_t)+e^{\alpha_t}\epsilon_t\ , \tag{8.10}$$

令式(8.10)为零可解得

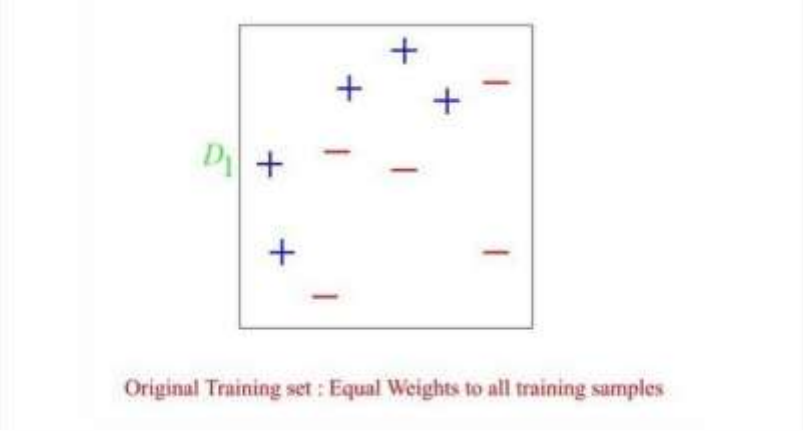
$$\alpha_t = \frac{1}{2}\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)\ , \tag{8.11}$$

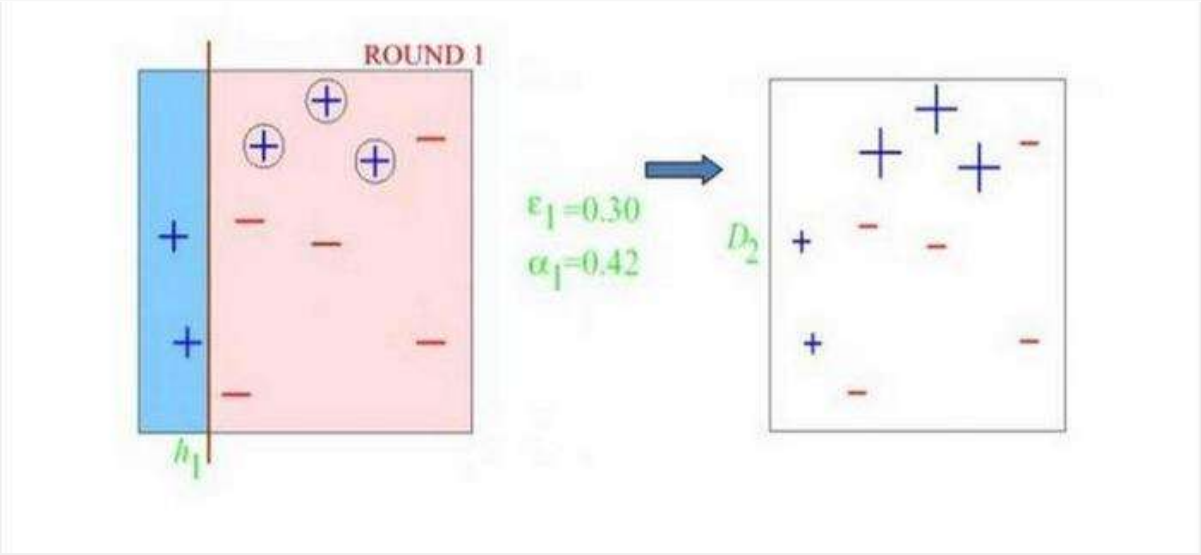
这恰是图 8.3 中算法第 6 行的分类器权重更新公式.

AdaBoost 算法在获得 $H_{t-1}$ 之后样本分布将进行调整, 使下一轮的基学习器 $h_t$ 能纠正 $H_{t-1}$ 的一些错误. 理想的 $h_t$ 能纠正 $H_{t-1}$ 的全部错误, 即最小化

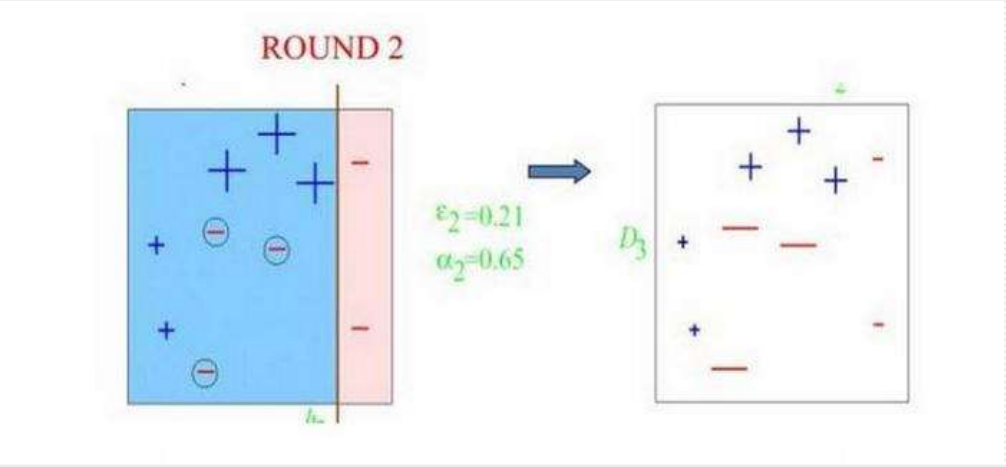
$$\begin{aligned}\ell_{\exp}(H_{t-1}+h_t \mid \mathcal{D}) &= \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}[e^{-f(\boldsymbol{x})(H_{t-1}(\boldsymbol{x})+h_t(\boldsymbol{x}))}] \\ &= \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}[e^{-f(\boldsymbol{x})H_{t-1}(\boldsymbol{x})}e^{-f(\boldsymbol{x})h_t(\boldsymbol{x})}]\ . \end{aligned} \tag{8.12}$$

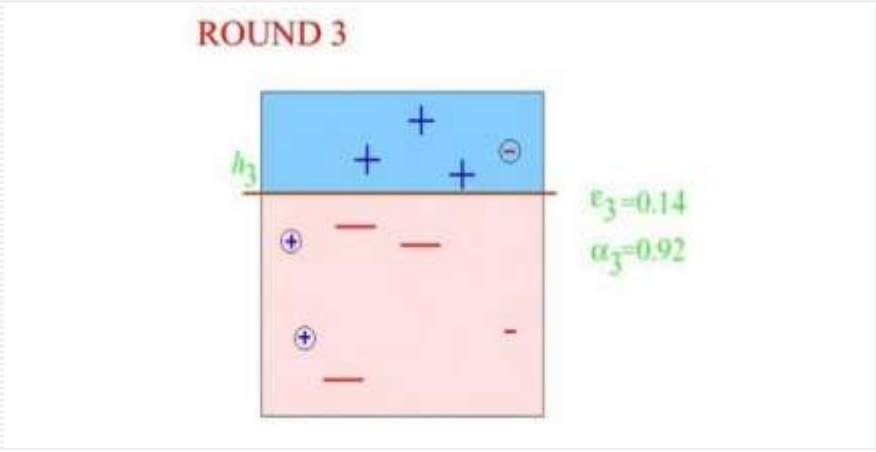
# 集成学习-Boosting

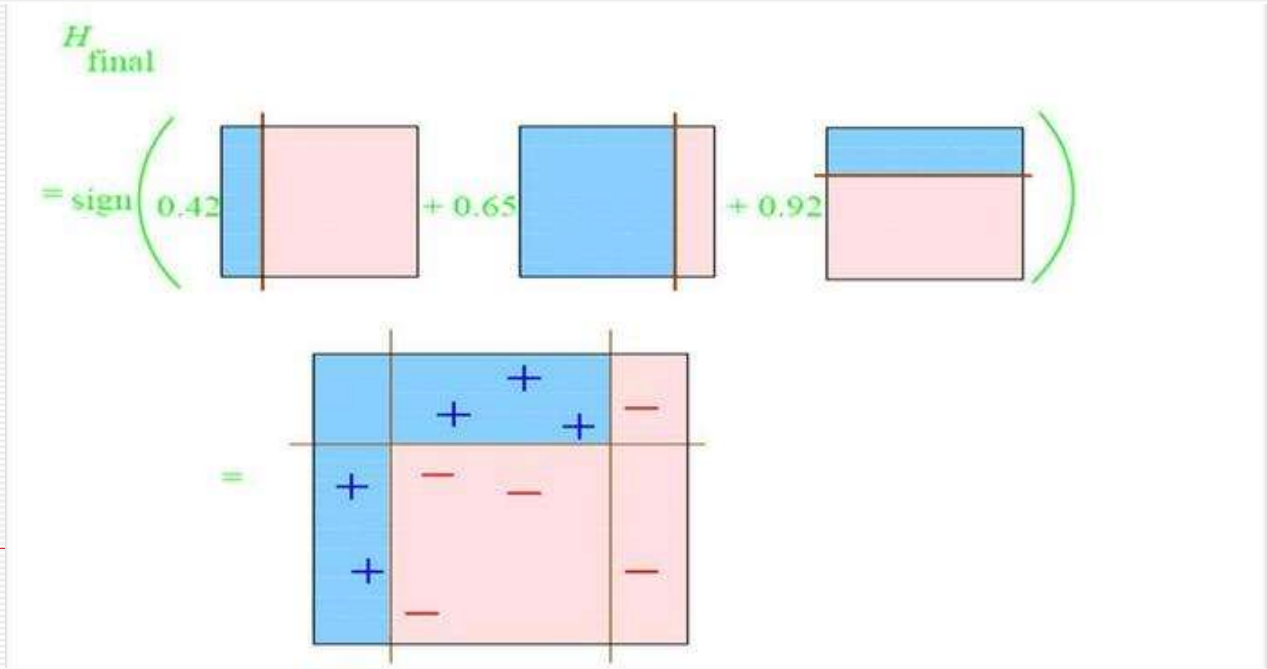


# 集成学习-Boosting



# 集成学习-Boosting

- 优点：
- 可以使用各种方法构造子分类器，Adaboost算法提供的是框架
- 简单，不用做特征筛选
- 相比较于RF，更不用担心过拟合问题

- **缺点：**
- adboost对于噪音数据和异常数据是十分敏感的。Boosting方法本身对噪声点异常点很敏感，因此在每次迭代时候会给噪声点较大的权重，这不是系统所期望的。
- 运行速度慢，凡是涉及迭代的基本上都无法采用并行计算，Adaboost是一种“串行”算法，所以GBDT(Gradient Boosting Decision Tree)也非常慢。

# 集成学习-Boosting

| 项目 | Bagging | Boosting |
| --- | --- | --- |
| **结构** | 并行 | 串行 |
| **训练集** | 独立 | 依赖 |
| **测试** | 可并行 | 需串行 |
| **作用** | 减小 variance | 减小 bias |

# 集成学习-Boosting

| 符号 | 涵义 |
| --- | --- |
| $\mathbf{x}$ | 测试样本 |
| $D$ | 数据集 |
| $y_D$ | $\mathbf{x}$ 在数据集中的标记 |
| $y$ | $\mathbf{x}$ 的真实标记 |
| $f$ | 训练集 $D$ 学得的模型 |
| $f(\mathbf{x};D)$ | 由训练集 $D$ 学得的模型 $f$ 对 $\mathbf{x}$ 的预测输出 |
| $\bar{f}(\mathbf{x})$ | 模型 $f$ 对 $\mathbf{x}$ 的 **期望预测** 输出 |

# 集成学习-Boosting

**泛化误差**

以回归任务为例，学习算法的平方预测误差期望为：

$$Err(\mathbf{x}) = E\left[(y - f(\mathbf{x}; D))^2\right]$$

**方差**

在一个训练集 $D$ 上模型 $f$ 对测试样本 $\mathbf{x}$ 的预测输出为 $f(\mathbf{x}; D)$，那么学习算法 $f$ 对测试样本 $\mathbf{x}$ 的 **期望预测** 为：

$$\bar{f}(\mathbf{x}) = E_D\left[f(\mathbf{x}; D)\right]$$

上面的期望预测也就是针对 **不同** 数据集 $D$，$f$ 对 $\mathbf{x}$ 的预测值取其期望

使用样本数相同的不同训练集产生的方差为：

$$var(\mathbf{x}) = E_D\left[\left(f(\mathbf{x}; D) - \bar{f}(\mathbf{x})\right)^2\right]$$

# 集成学习-Boosting

## 噪声

噪声为真实标记与数据集中的实际标记间的偏差:

$$\epsilon^2 = E_D\left[(y_D - y)^2\right]$$

## 偏差

期望预测与真实标记的误差称为偏差(bias), 为了方便起见, 我们直接取偏差的平方:

$$bias^2(\mathbf{x}) = \left(\overline{f}(\mathbf{x}) - y\right)^2$$

# 集成学习-Boosting

对算法的期望泛化误差进行分解:

$$
\begin{aligned}
E(f;D) =& \mathbb{E}_D\left[(f(\boldsymbol{x};D)-y_D)^2\right] \\
=& \mathbb{E}_D\left[(f(\boldsymbol{x};D)-\bar{f}(\boldsymbol{x})+\bar{f}(\boldsymbol{x})-y_D)^2\right] \\
=& \mathbb{E}_D\left[(f(\boldsymbol{x};D)-\bar{f}(\boldsymbol{x}))^2\right]+\mathbb{E}_D\left[(\bar{f}(\boldsymbol{x})-y_D)^2\right] \\
&+\mathbb{E}_D\left[2\left(f(\boldsymbol{x};D)-\bar{f}(\boldsymbol{x})\right)\left(\bar{f}(\boldsymbol{x})-y_D\right)\right] \\
=& \mathbb{E}_D\left[(f(\boldsymbol{x};D)-\bar{f}(\boldsymbol{x}))^2\right]+\mathbb{E}_D\left[(\bar{f}(\boldsymbol{x})-y_D)^2\right] \\
=& \mathbb{E}_D\left[(f(\boldsymbol{x};D)-\bar{f}(\boldsymbol{x}))^2\right]+\mathbb{E}_D\left[(\bar{f}(\boldsymbol{x})-y+y-y_D)^2\right] \\
=& \mathbb{E}_D\left[(f(\boldsymbol{x};D)-\bar{f}(\boldsymbol{x}))^2\right]+\mathbb{E}_D\left[(\bar{f}(\boldsymbol{x})-y)^2\right]+\mathbb{E}_D\left[(y-y_D)^2\right] \\
&+2\mathbb{E}_D\left[(\bar{f}(\boldsymbol{x})-y)(y-y_D)\right] \\
=& \mathbb{E}_D\left[(f(\boldsymbol{x};D)-\bar{f}(\boldsymbol{x}))^2\right]+(\bar{f}(\boldsymbol{x})-y)^2+\mathbb{E}_D\left[(y-y_D)^2\right]
\end{aligned}
$$

$$
Err(\boldsymbol{x}) = \mathbb{E}_D\left[(f(\boldsymbol{x};D)-\bar{f}(\boldsymbol{x}))^2\right] + (\bar{f}(\boldsymbol{x})-y)^2 + \mathbb{E}_D\left[(y_D-y)^2\right]
$$

- variance
- bias$^2$
- noise

# 集成学习-Boosting

- 偏差.

  偏差度量了学习算法的期望预测与真实结果的偏离程序, 即 **刻画了学习算法本身的拟合能力** .

- 方差.

  方差度量了同样大小的训练集的变动所导致的学习性能的变化, 即 **刻画了数据扰动所造成的影响** .

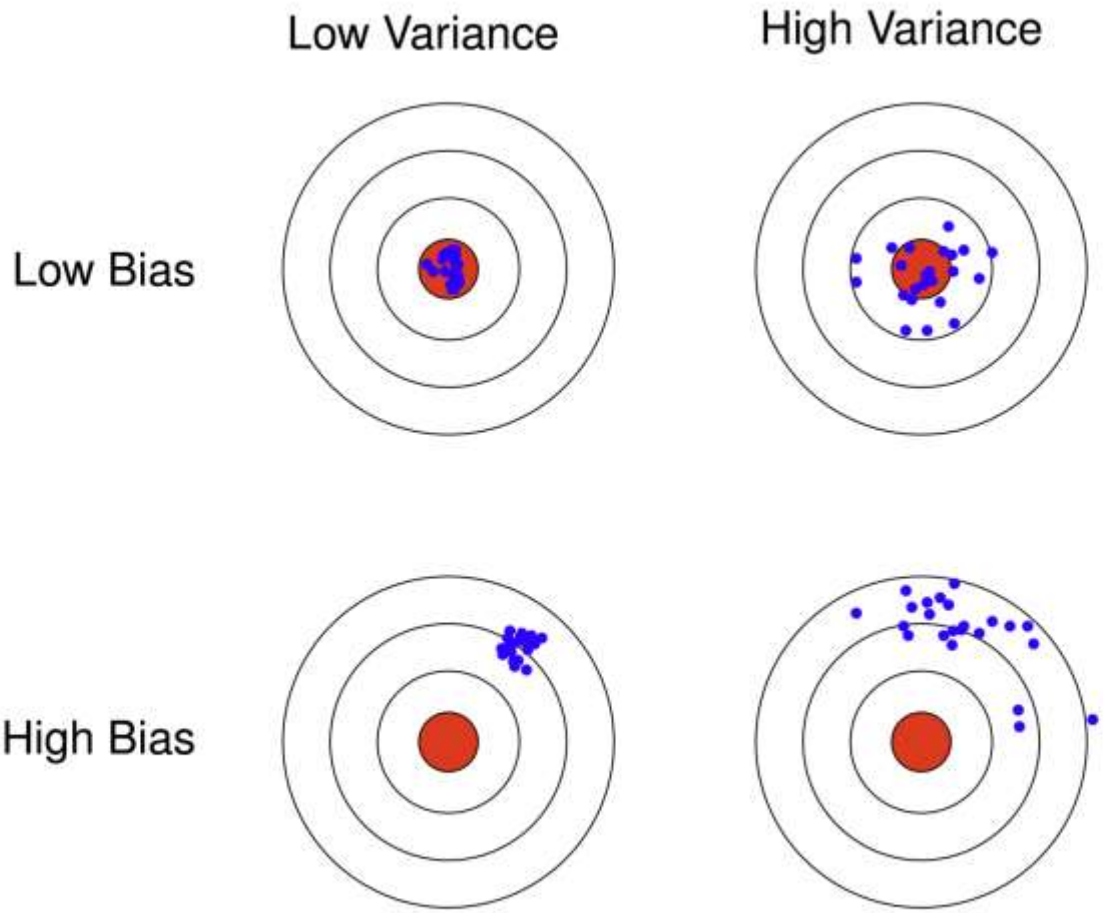
- 噪声.

  噪声表达了在当前任务上任何学习算法所能达到的期望泛化误差的下界, 即 **刻画了学习问题本身的难度** . 巧妇难为无米之炊, 给一堆很差的食材, 要想做出一顿美味, 肯定是很有难度的.

想当然地, 我们希望偏差与方差越小越好, 但实际并非如此. 一般来说, 偏差与方差是有冲突的, 称为偏差-方差窘境 (bias-variance dilemma).

- 给定一个学习任务, 在训练初期, 由于训练不足, 学习器的拟合能力不够强, 偏差比较大, 也是由于拟合能力不强, 数据集的扰动也无法使学习器产生显著变化, 也就是欠拟合的情况;
- 随着训练程度的加深, 学习器的拟合能力逐渐增强, 训练数据的扰动也能够渐渐被学习器学到;
- 充分训练后, 学习器的拟合能力已非常强, 训练数据的轻微扰动都会导致学习器发生显著变化, 当训练数据自身的、非全局的特性被学习器学到了, 则将发生过拟合.

# 集成学习-Boosting



# 集成学习-CART回归树

## 算法思想

CART算法采用的是一种二分递归分割的技术，将当前样本分成两个子样本集，使得生成的非叶子节点都有两个分支。因此CART实际上是一颗二叉树。

## CART树的特点

- CART是一颗二叉树
- CART既是分类树又是回归树
- 当CART是分类树的时候，采用GINI值作为分裂节点的依据，当CART作为回归树的时候，使用样本的最小方差作为分裂节点的依据

# 集成学习-CART回归树

**最小二乘法回归树生成算法**

输入：训练数据集D

输出：回归树$f(x)$

在训练数据集所在的输入空间中，递归得将每一个区域划分为两个子区域并决定每个子区域上的输出值，构建二叉决策树：

(1)选择最优切分变量j和切分点s，求解

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

遍历变量j，对固定的切分变量j扫描切分点s，选择使上式达到误差最小的变量(j,s)，其中$R_1$和$R_2$表示的是划分之后的空间。

(2)用选定的(j,s)划分区域并决定响应的输出值。

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

$$c_m=\frac{1}{N_m}\sum_{x_i\in R_m(j,s)}y_i,\quad x\in R_m,m=1,2$$

(3)继续对两个子区域调用步骤(1),(2),直到满足停止条件。

(4)将输入空间划分为M个区域$R_1,R_2,R_3....R_M$，生成决策树：

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

# 集成学习-CART回归树

## 分类树的生成

分类树是用基尼指数选择最优特征，同时决定该特征的最优二值切分点。

### 基尼指数

分类问题中，假设有K个类，样本点属于第K类的概率为$p_k$，则概率分布的基尼指数定义为

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

对于二分类问题来说，若样本点属于第一类的概率为p，则概率分布的基尼指数为

$$Gini(p) = 2p(1-p)$$

对于给定的样本集合D，其基尼指数为

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

其中，$C_k$是D中属于第k类的样本子集，K是类的个数。$|C_k|$和$D$分别表示子集的个数和样本的个数。

如果样本集合D根据特征A是否取某一可能的值$\alpha$被分割成$D_1$和$D_2$，即

$$D_1 = \{(x,y) \in D | A(x) = a\}, \quad D_2 = D - D_1$$

所以在特征A的条件下集合D的基尼指数为

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

其中基尼指数Gini(D)表示集合的不确定性，基尼指数G(D,A)表示A=a分解后集合的不决定性。基尼指数越大，样本集合的不确定性越大。

# 集成学习-CART回归树

**分类树生成算法**

输入：训练数据集D,停止计算的条件

输出：CART决策树

具体步骤:

(1)计算现有特征对该数据集的基尼指数，对于每一个特征A，可以对样本点A是否为a可以将数据集D分成数据集$D_1, D_2$。

(2)对于所有的特征A和所有可能的切分点a，选择基尼指数最小的特征以及相对应的切分点作为最优特征和最佳切分点。

(3)对最优子树递归调用(1)(2)，直到满足停止条件。

(4)生成CART分类树。

# 集成学习-CART回归树

**算法 5.5（最小二乘回归树生成算法）**

输入：训练数据集 $D$；

输出：回归树 $f(x)$.

在训练数据集所在的输入空间中，递归地将每个区域划分为两个子区域并决定每个子区域上的输出值，构建二叉决策树：

（1）选择最优切分变量 $j$ 与切分点 $s$，求解

$$\min_{j,s}\left[\min_{c_1}\sum_{x_i\in R_1(j,s)}(y_i-c_1)^2+\min_{c_2}\sum_{x_i\in R_2(j,s)}(y_i-c_2)^2\right] \tag{5.21}$$

遍历变量 $j$，对固定的切分变量 $j$ 扫描切分点 $s$，选择使式 (5.21) 达到最小值的对 $(j,s)$.

（2）用选定的对 $(j,s)$ 划分区域并决定相应的输出值：

$$R_1(j,s)=\{x\mid x^{(j)}\leqslant s\},\quad R_2(j,s)=\{x\mid x^{(j)}>s\}$$

$$\hat{c}_m=\frac{1}{N_m}\sum_{x_i\in R_m(j,s)}y_i,\quad x\in R_m,\quad m=1,2$$

（3）继续对两个子区域调用步骤 (1)，(2)，直至满足停止条件.

（4）将输入空间划分为 $M$ 个区域 $R_1,R_2,\cdots,R_M$，生成决策树：

$$f(x)=\sum_{m=1}^{M}\hat{c}_m I(x\in R_m)$$

■

# 集成学习-CART回归树

表 8.2　训练数据表

| $x_i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| $y_i$ | 5.56 | 5.70 | 5.91 | 6.40 | 6.80 | 7.05 | 8.90 | 8.70 | 9.00 | 9.05 |

**解**　按照算法 8.3，第 1 步求 $f_1(x)$ 即回归树 $T_1(x)$.

首先通过以下优化问题：

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

求解训练数据的切分点 $s$：

$$R_1=\{x\mid x\leqslant s\},\quad R_2=\{x\mid x>s\}$$

容易求得在 $R_1$，$R_2$ 内部使平方损失误差达到最小值的 $c_1$，$c_2$ 为

$$c_1=\frac{1}{N_1}\sum_{x_i\in R_1}y_i,\quad c_2=\frac{1}{N_2}\sum_{x_i\in R_2}y_i$$

这里 $N_1$，$N_2$ 是 $R_1$，$R_2$ 的样本点数.

求训练数据的切分点. 根据所给数据，考虑如下切分点：

$$1.5,\ 2.5,\ 3.5,\ 4.5,\ 5.5,\ 6.5,\ 7.5,\ 8.5,\ 9.5$$

对各切分点，不难求出相应的 $R_1$，$R_2$，$c_1$，$c_2$ 及

$$m(s)=\min_{c_1}\sum_{x_i\in R_1}(y_i-c_1)^2+\min_{c_2}\sum_{x_i\in R_2}(y_i-c_2)^2$$

例如，当 $s=1.5$ 时，$R_1=\{1\}$，$R_2=\{2,3,\cdots,10\}$，$c_1=5.56$，$c_2=7.50$，

$$m(s)=\min_{c_1}\sum_{x_i\in R_1}(y_i-c_1)^2+\min_{c_2}\sum_{x_i\in R_2}(y_i-c_2)^2=0+15.72=15.72$$

现将 $s$ 及 $m(s)$ 的计算结果列表如下（见表 8.3）.

# 集成学习-CART回归树

表 8.3 计算数据表

| $s$ | 1.5 | 2.5 | 3.5 | 4.5 | 5.5 | 6.5 | 7.5 | 8.5 | 9.5 |
|---|---|---|---|---|---|---|---|---|---|
| $m(s)$ | 15.72 | 12.07 | 8.36 | 5.78 | 3.91 | 1.93 | 8.01 | 11.73 | 15.74 |

由表 8.3 可知，当$s=6.5$时$m(s)$达到最小值，此时$R_1=\{1,2,\cdots,6\}$，$R_2=\{7,8,9,10\}$，$c_1=6.24$，$c_2=8.91$，所以回归树$T_1(x)$为

$$T_1(x)=\begin{cases}6.24, & x<6.5\\ 8.91, & x\geqslant 6.5\end{cases}$$

$$f_1(x)=T_1(x)$$

# 常见优化方法

大部分的机器学习算法的本质都是建立优化模型，通过最优化方法对目标函数（或损失函数）进行优化，从而训练出最好的模型。常见的最优化方法有梯度下降法、牛顿法和拟牛顿法、共轭梯度法等。

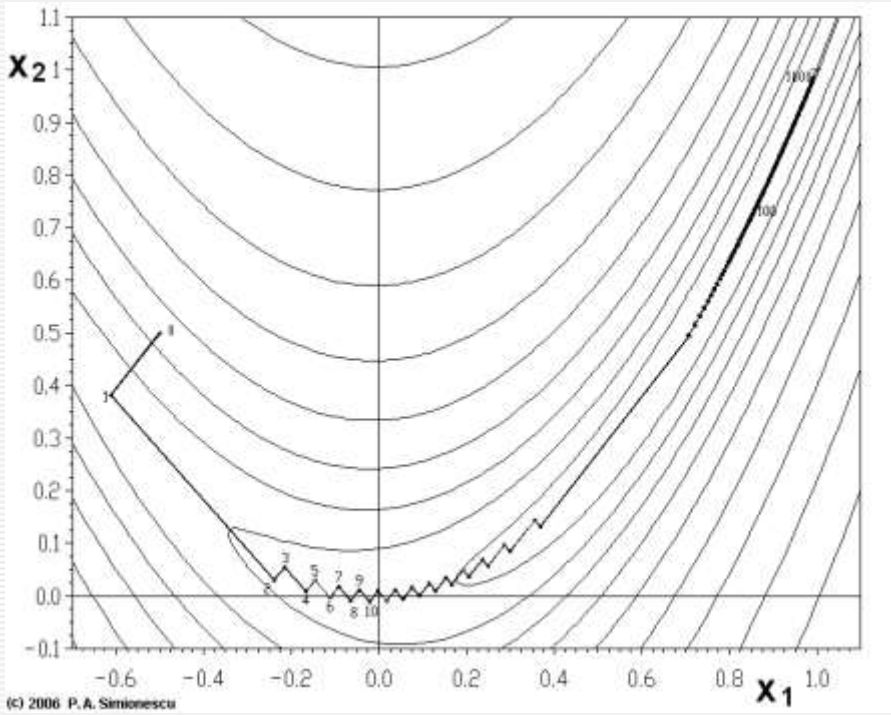
**1. 梯度下降法（Gradient Descent）**

梯度下降法实现简单，当目标函数是凸函数时，梯度下降法的解是全局解。一般情况下，其解不保证是全局最优解，梯度下降法的速度也未必是最快的。**梯度下降法的优化思想是用当前位置负梯度方向作为搜索方向，因为该方向为当前位置的最快下降方向，所以也被称为是”最速下降法“。最速下降法越接近目标值，步长越小，前进越慢。**

# 常见优化方法

**梯度下降法的缺点：**
**（1）靠近极小值时收敛速度减慢，如图所示；**
**（2）直线搜索时可能会产生一些问题；**
**（3）可能会“之字形”地下降。**

从图可以看出，梯度下降法在接近最优解的区域收敛速度明显变慢，利用梯度下降法求解需要很多次的迭代。
在机器学习中，基于基本的梯度下降法发展了两种梯度下降方法，分别为随机梯度下降法和批量梯度下降法。



# 常见优化方法

## 2. 牛顿法（Newton's method）

牛顿法的基本思想是：在现有极小点估计值的附近对$f(x)$做二阶泰勒展开，进而找到极小点的下一个估计值。设$x_k$为当前的极小点估计值，则：

$$\varphi(x) = f(x_k) + f'(x_k)(x - x_k) + \frac{1}{2}f''(x_k)(x - x_k)^2$$

表示$f(x)$在$x_k$附近的二阶泰勒展开式（略去了关于$x - x_k$的高阶项），由于求的是最值，由极值必要条件可知，$\varphi(x)$应满足

$$\varphi'(x) = 0$$

即

$$f'(x_k) + f''(x_k)(x - x_k) = 0$$

从而得到

$$x = x_k - \frac{f'(x_k)}{f''(x_k)}$$

于是，若给定初始值$x_0$,则可以构造如下的迭代格式

$$x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)}, k = 0, 1....$$

# 常见优化方法

对于N>1的情形，二阶泰勒展开式可以做推广，此时

$$\varphi(X) = f(X_k) + \bigtriangledown f(X_k)(X - X_k) + \frac{1}{2}(X - X_k)^T \bigtriangledown^2 f(X_k)(X - X_k)$$

其中$\bigtriangledown f$为$f$的梯度向量，$\bigtriangledown^2 f$为$f$的海森矩阵，其定义如下：

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_N} \end{bmatrix}, \quad \nabla^2 f = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_N} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_N} \\ & & \ddots & \\ \frac{\partial^2 f}{\partial x_N \partial x_1} & \frac{\partial^2 f}{\partial x_N \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_N^2} \end{bmatrix}_{N \times N}$$

# 常见优化方法

注意，$\bigtriangledown f$和$\bigtriangledown^2 f$中的元素均为关于$X$的函数，以下简记为$g$和$H$.特别是若$f$的混合偏导数可交换次序，则海森矩阵为对称矩阵。而$\bigtriangledown f(X_k)$和$\bigtriangledown^2 f(X_k)$则表示将$X$取为$x_k$后得到的实值向量和矩阵，以下分别将其简记为$g_k$和$H_k$

同样的，由于是求极小点，极值必要条件 要求它为$\varphi(X)$的驻点，即

$$\bigtriangledown \varphi(X) = 0$$

通过在上式中两边作用一个梯度算子，得到：

$$g_k + H_k(X - X_k) = 0$$

此时若矩阵$H_k$非奇异，可解得

$$X = X_k - H_k^{-1} g_k$$

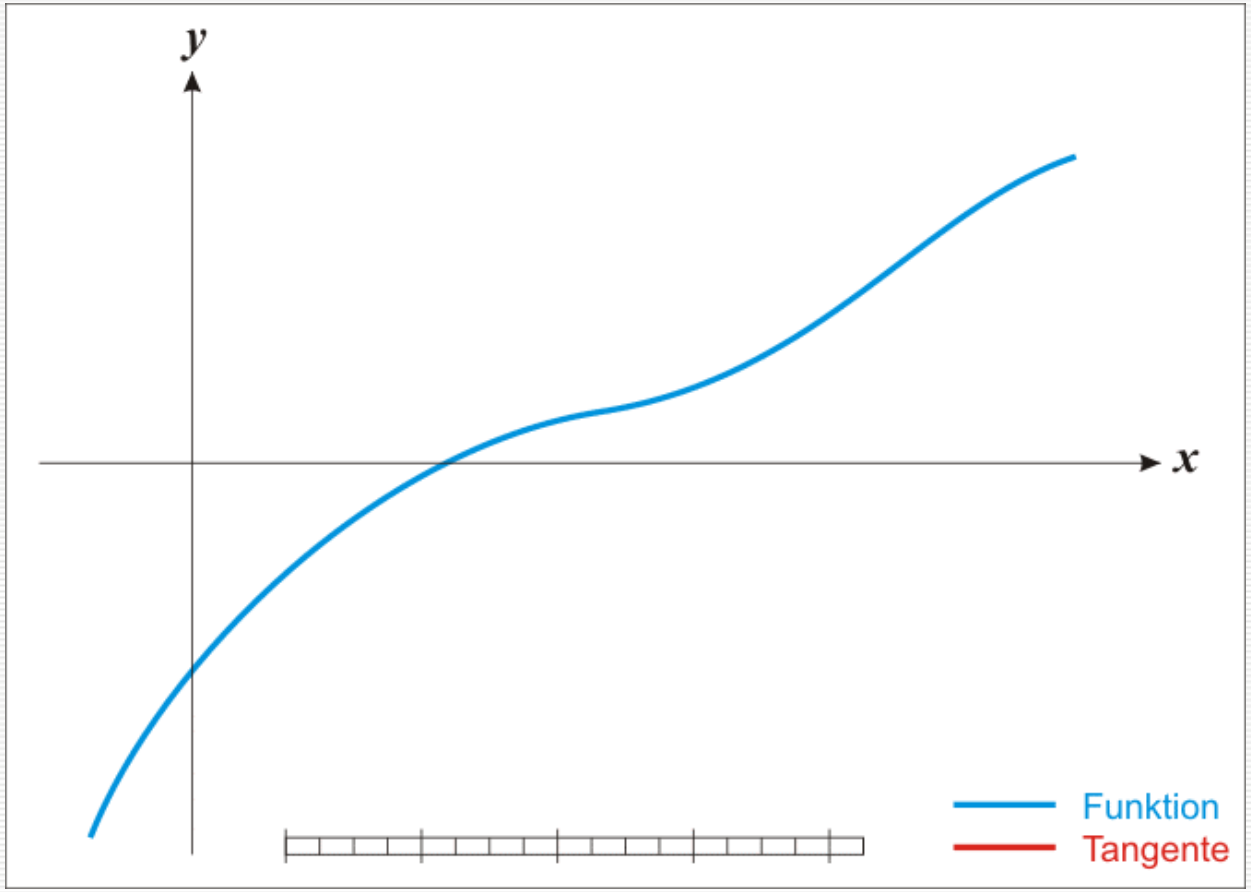
若给定初始值$X_0$,则可同样构造出迭代格式

$$X_{k+1} = X_k - H_k^{-1} g_k$$

这就是牛顿迭代法，其迭代格式中的搜索方向为$-H_k^{-1} g_k$称为牛顿方向。

当目标函数是二次函数时，由于二次泰勒展开函数与原目标函数不是近似而是完全相同的二次式，海森矩阵退化为一个常数矩阵，从任一初始点出发，只需一步迭代即可达到极小点，因此牛顿法是一种具有二次收敛的算法。对于非二次函数，若函数的二次性态较强，或迭代点已进入极小点的邻域，则其收敛速度也是很快的。

# 常见优化方法




牛顿法的优缺点总结：

优点：二阶收敛，收敛速度快；

缺点：牛顿法是一种迭代算法，每一步都需要求解目标函数的**Hessian**矩阵的逆矩阵，计算比较复杂。

# 常见优化方法

**3. 拟牛顿法（Quasi-Newton Methods）**

**拟牛顿法的本质思想是改善牛顿法每次需要求解复杂的Hessian矩阵的逆矩阵的缺陷，它使用正定矩阵来近似Hessian矩阵的逆，从而简化了运算的复杂度。**拟牛顿法和最速下降法一样只要求每一步迭代时知道目标函数的梯度。通过测量梯度的变化，构造一个目标函数的模型使之足以产生超线性收敛性。这类方法大大优于最速下降法，尤其对于困难的问题。另外，因为拟牛顿法不需要二阶导数的信息，所以有时比牛顿法更为有效。

BFGS(Broyden–Fletcher–Goldfarb–Shanno)的算法流程如下：

1. 初始化：初始点$x_0$以及近似逆Hessian矩阵$B_0^{-1}$。通常，$B_0 = I$,既为单位矩阵。
2. 计算线搜索方向：$\mathbf{p}_k = -B_k^{-1}\nabla f(\mathbf{x}_k)$
3. 用"Backtracking line search"算法沿搜索方向找到下一个迭代点：$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$
4. 根据**Armijo–Goldstein** 准则，判断是否停止。
5. 计算$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$;以及 $\mathbf{y}_k = \nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k)$
6. 迭代近似逆Hessian矩阵：

$$B_{k+1}^{-1} = \left(I - \frac{s_k y_k^T}{y_k^T s_k}\right) B_k^{-1} \left(I - \frac{y_k s_k^T}{y_k^T s_k}\right) + \frac{s_k s_k^T}{y_k^T s_k}$$

# 常见优化方法

**4. 共轭梯度法（Conjugate Gradient）**

对于对称正定矩阵Q的n元二次函数，根据共轭方向的性质，依次沿着对Q共轭的一组方向作一维搜索，则可保证在至多n步内获得二次函数的极小点。共轭方向法在处理非二次目标函数时也相当有效，具有超线性的收敛速度，在一定程度上克服了最速下降法的锯齿形现象，同时又避免了牛顿法所涉及的**Hessian**矩阵的计算和求逆问题。对于非二次函数，n步搜索并不能获得极小点，需采用重开始策略，即在每进行n次一维搜索之后，若还未获得极小点，则以负梯度方向作为初始方向重新构造共轭方向，继续搜索。

在各种优化算法中，共轭梯度法是非常重要的一种。其优点是所需存储量小，具有步收敛性，稳定性高，而且不需要任何外来参数。

设Q为对称正定矩阵，若一组非零向量S1，S2，…Sn满足

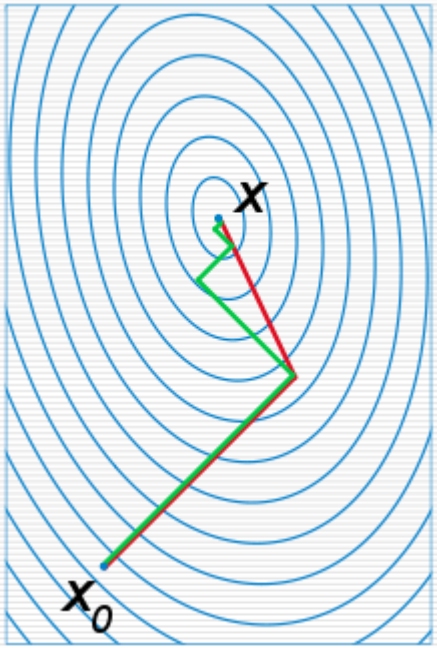
Si Q Sj=0 （i≠j） （1）

则称向量系Si（i=1,2，…n）为关于矩阵Q共轭。

共扼向量的方向称为共轭方向。

如果Q为单位矩阵，则式(1)即成为Si Sj，这样两个向量的点积（或称内积）为零，此二向量在几何上是正交的，它是共轭的一种特例。

# 常见优化方法




梯度下降在寻找搜索方向的时候只利用了空间中当前点的信息，共轭梯度下降还利用了之前的搜索路径信息。

共轭梯度法是共轭方向法的一种，只需要知道上一步的的共轭向量以及梯度方向，就可以计算出下一个共轭方向。

$$p_k = -r_k + \beta_k p_{k-1}$$

所以在存储空间和计算上有优势。

绿色为梯度下降法，红色代表共轭梯度法

# 谢谢