第六章 决策树，随机森林与梯度提升树

1 决策树

1 .1 什么是决策树

一个女孩的母亲要给这个女孩介绍男朋友，于是有了下面的对话：

女儿：多大年纪了？

母亲：26。

女儿：长的帅不帅？

母亲：挺帅的。

女儿：收入高不？

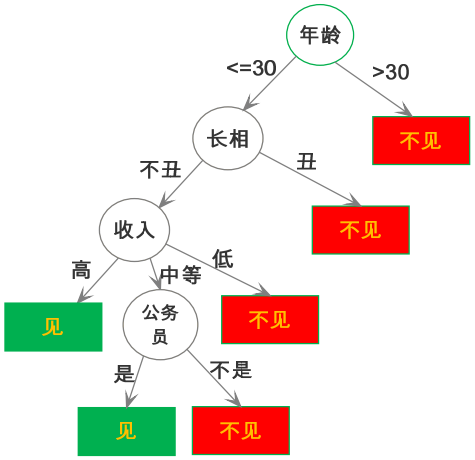
母亲：不算很高，中等情况。

女儿：是公务员不？

母亲：是，在税务局上班呢。

女儿：那好，我去见见。

决策过程如右图。

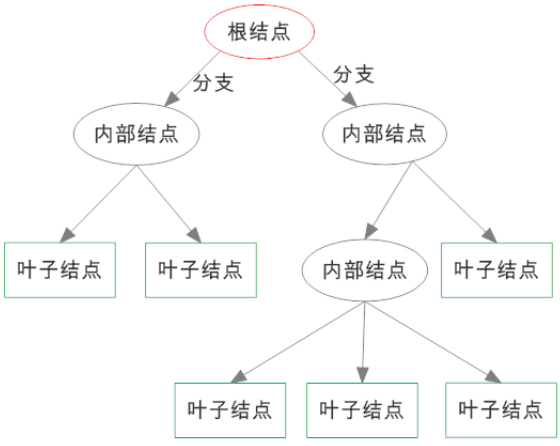


这个女孩的决策过程就是典型的分类树决策。相当于通过年龄、长相、收入和是否公务员对将男人分为两个类别：见和不见。

决策树（Decision Tree），是用于分类和预测的一种树结构，由**结点**和**分支**组成。决策树学习是以实例为基础的归纳学习算法。它着眼于从一组无次序、无规则的实例中推理出决策树表示形式的分类规则。结点有两种类型：**内部结点**和**叶结点**。内部结点表示一个特征，叶子结点表示一个类。

用决策树分类，从根结点开始，对实例的某一特征进行测试，根据测试结果，将实例分配到其子结点；这时，每一个子结点对应着该特征的一个取值，如此递归地对实例进行测试并分配，直至到达叶结点。最后将实例分到叶结点的类中。

下图是一个决策树的示意图。



可以将决策树看成一个 **if-then 规则**的集合。规则：由决策树的根结点的每一条路径构建一条规则；路径上内部结点的特征对应规则的条件，而叶结点的类对应着规则的结论。决策树的路径或其对应的 if-then 规则集合具有一个重要的性质：**互斥并且完备**。即每一个实例都被一条路径或者一条规则所覆盖，而且只被一条路径或一条规则所覆盖。

对于相亲决策树可提取以下分类规则：

IF 年龄= ‘>30’ THEN 见面意向= ‘不见’；

IF 年龄= ‘<=30’ AND 长相= ‘丑’ THEN 是否见面意向= ‘不见’；

IF 年龄= ‘<=30’ AND 长相= ‘不丑’ AND 收入= ‘低’ THEN 见面意向= ‘不见’；

IF 年龄= ‘<=30’ AND 长相= ‘不丑’ AND 收入= ‘高’ THEN 见面意向= ‘见’；

IF 年龄= ‘<=30’ AND 长相= ‘不丑’ AND 收入= ‘中等’ AND 公务员= ‘是’ THEN 见面意向= ‘见’；

IF 年龄= ‘<=30’ AND 长相= ‘不丑’ AND 收入= ‘中等’ AND 公务员= ‘否’ THEN 见面意向= ‘不见’；

决策树含义直观,容易解释。对于实际应用,决策树还有其他算法难以比肩的速度优势.这使得决策树,一方面能够有效地进行大规模数据的处理和学习;另一方面在测试/预测阶段满足实时或者更高的速度要求。历史上,因为预测结果方差大而且容易过拟合,决策树曾经一度被学术界冷落。但是在近十年，随着集成学习(Ensemble Learning)的发展和大数据时代的到来，决策树的缺点被逐渐克服，同时它的优点得到了更好的发挥。在工程界，决策树以及对应的集成学习算法(如随机森林)已经成为解决实际问题的重要工具之一。

1.2 决策树构建与学习

构建决策树采用自上而下的递归构造方法。以多叉树为例，构造思路是：如果训练数据集中所有数据是**同类的**，则将之作为叶子结点，结点内容即是该类别标记，否则根据**某种策略**选择一个特征，按照特征的各个取值，把数据集合划分若干个子集，使得每个子集上的所有数据在该特征上具有同样的特征值 ，然后再依次递归处理各个子集。这种思路称之为“**分而治之**”。

决策树学习本质上是从训练数据集中归纳出一组分类规则。决策树学习的算法通常是一个递归的选择最优特征，并根据该特征对训练数据进行分割，使得对各子数据集有一个最好的分类的过程。这一过程对应着对特征空间的划分，也对应着决策树的构建。

a. 构建根结点，所有训练数据放在根结点

b. 选择一个**最优特征**，按照这一特征将训练数据集分割成子集，使得各个子集有一个在当前条件下的最好分类

c. 若这些子集已经能够被基本分类，那么构建叶结点，并将这些子集分到所对应的叶结点中去。

d. 如果还有子集不能被正确分类，那么就对这些子集选择新的最优特征，继续对其进行分割，构建相应结点

e. 如此递归下去，直至所有训练数据子集被基本正确分类，或者没有合适的特征为止。最后每个子集都被分到叶结点上，即都有了明确的类。这就生成了一棵决策树。

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。

决策树学习常用的算法有 ID3、C4.5 与 CART。

1.3 特征选择

决策树的特征可以有两种：

1） 数字型（Numeric）：特征类型是整数或浮点数，如前面例子中的“年收入”。用“>=”，“>”，“<”或“<=”作为划分条件。

2） 名称型（Nominal）：特征在有限的选项中取值，比如前面例子中的“收入”，只能是“高”，“中等”或“低”。使用“=”来划分。

在决策树的构建过程中，需要按照一定的次序从全部的特征中选取特征。**待选特征**就是在目前的步骤之前还没有被选择的特征的集合。例如，全部的特征是 ABCDE，第一步的时候，待选特征就是 ABCDE，第一步选择了 C，那么第二步的时候，待选特征就是 ABDE。

接待选特征的定义，每一次选取的特征就是**分裂特征**，例如，在上面的例子中，第一步的分裂特征就是 C。因为选出的这些特征将数据集分成了一个个不相交的部分，所以叫它们分裂特征。

选取对训练数据具有分类能力的特征，用该特征来划分特征空间。 如果一个特征具有更好的分类能力，或者说，按照这一特征将训练数据集分割成子集，使得各个子集的“**纯度**”更高，那么就更应该选择这个特征。通常特征选择的准则是信息增益或信息增益率。我们首先回顾熵的有关概念。

熵是表示随机变量不确定性的度量。设 X 是一个取有限个值的离散随机变量，其概率分布为

$$P\left(X=x_i\right)=p_i,\quad i=1,2,...,n$$

则随机变量 X 的熵定义为

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

熵越大，随机变量的不确定性越大。

条件熵 $H\left(Y\mid X\right)$ 表示在已知随机变量 X 的条件下随机变量 Y 的不确定性。条件熵 $H\left(Y\mid X\right)$ 定义为 X 给定条件下 Y 的条件概率分布的熵对 X 的数学期望

$$H(Y\mid X)=\sum_{i=1}^{m}p_iH(Y\mid X=x_i)$$

通常，$H\left(Y\right)-H\left(Y\mid X\right)$ 称 X 和 Y 的**互信息**，其意义为因为 X 的出现导致 Y 的不确定性减少。

当熵与条件熵中概率由数据估计得到时，所对应的熵和条件熵称为经验熵和经验条件熵。

设训练数据集为 D，样本总数为 $\left|D\right|$，其中包含 K 个不同的类 C_k（$k=1,2,...,K$）。$\left|C_k\right|$ 是 D 中属于类 C_k 的样品的个数。数据集 D 的**经验熵**为

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

其中 p_k 是样本属于类 C_k 的概率，用 $\left|C_k\right|/\left|D\right|$ 来估计。即

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

经验熵 $H\left(D\right)$ 表示对数据集 D 进行分类的不确定性。

设特征 A 有 n 个不同的取值，根据特征 A 的取值将数据集 D 划分为 n 个子集 D_1，

D_2 ，...，D_n，$|D_i|$ 为 D_i 的样本个数。记子集 D_i 中属于类 C_k 的样本的集合为 D_{ik} ，即 $D_{ik}=D_i\cap C_k$ ，$|D_{ik}|$ 为 D_{ik} 的样本个数。特征 A 对数据集 D 的**经验条件熵**为

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

特征 A 对训练数据集 D 的**信息增益**（information gain）$g(D,A)$，定义为集合 D 的经验熵 $H(D)$ 与特征 A 给定条件下的经验条件熵 $H(D|A)$ 之差，即

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

从定义来看，信息增益等价于数据集中分类类别与特征的互信息。

决策树 ID3 算法应用信息增益准则选择特征。给定训练数据集和特征 A，经验熵 $H(D)$ 表示对数据集 D 进行分类的不确定性，而经验条件熵 $H(D|A)$ 表示在特征 A 给定的条件下对数据集 D 进行分类的不确定性。那么它们的差，即信息增益，就表示由于特征 A 的信息使得对数据集 D 的分类的不确定性减少的程度。显然，对于数据集 D 而言，信息增益依赖于特征，不同的特征往往具有不同的信息增益，信息增益大的特征具有更强的分类能力。

给定数据集 D 或其子集 D_i，计算每个特征的信息增益，并比较他们的大小，**选择信息增益最大的特征作为最优特征**。

下面通过例子来说明如何根据信息增益大小选择最优特征。

下表是一个由 15 个样本组成的贷款申请训练数据。数据包括贷款申请人的 4 个特征（属性）：年龄、有工作、有自己的房子、信贷情况，每个特征有不同的取值，如年龄有老、中、青 3 种取值。表的最后一列是类别，分为是否贷款 2 个类。

我们的目的是构建决策树，使得计算机对贷款申请人员的申请信息自动进行分类，以决定能否贷款。

ID	年龄	有工作	有自己的房子	信贷情况	类别
1	青年	否	否	一般	否
2	青年	否	否	好	否
3	青年	是	否	好	是
4	青年	是	是	一般	是
5	青年	否	否	一般	否
6	中年	否	否	一般	否
7	中年	否	否	好	否
8	中年	是	是	好	是
9	中年	否	是	非常好	是
10	中年	否	是	非常好	是
11	老年	否	是	非常好	是
12	老年	否	是	好	是
13	老年	是	否	好	是
14	老年	是	否	非常好	是
15	老年	否	否	一般	否

首先计算经验熵，15 个样本中，9 个“是”，6 个“否”，所以

$$H(D)=-\frac{9}{15}\log_2\frac{9}{15}-\frac{6}{15}\log_2\frac{6}{15}=0.971$$

然后计算各特征对数据集 D 的信息增益。

首先计算经验条件熵 $H(D|\text{年龄})$。年龄的取值为青年、中年、老年。

青年共 5 个样本 (D_1)，其中类别为“是”2 个样本，“否”3 个样本；

中年共 5 个样本 (D_2)，其中类别为“是”3 个样本，“否”2 个样本；

老年共 5 个样本 (D_3)，其中类别为“是”4 个样本，“否”1 个样本；

计算特征“年龄”为条件的的经验条件熵 $H(D|\text{年龄})$：

$$\begin{aligned}H(D|\text{年龄})&=\frac{5}{15}H(D_1)+\frac{5}{15}H(D_2)+\frac{5}{15}H(D_3)\\&=\frac{5}{15}\left(-\frac{2}{5}\log_2\frac{2}{5}-\frac{3}{5}\log_2\frac{3}{5}\right)+\frac{5}{15}\left(-\frac{3}{5}\log_2\frac{3}{5}-\frac{2}{5}\log_2\frac{2}{5}\right)+\frac{5}{15}\left(-\frac{4}{5}\log_2\frac{4}{5}-\frac{1}{5}\log_2\frac{1}{5}\right)\\&=0.888\end{aligned}$$

得到经验条件熵 $H(D|\text{年龄})$ 为

$$\begin{aligned}g(D,\text{年龄})&=H(D)-H(D|\text{年龄})\\&=0.971-0.888=0.083\end{aligned}$$

类似可以计算:

特征“有工作”的信息增益

$$\begin{aligned}g(D,\text{有工作})&=H(D)-H(D|\text{有工作})\\&=0.971-\left[\frac{5}{15}\times 0+\frac{10}{15}\left(-\frac{4}{10}\log_2\frac{4}{10}-\frac{6}{10}\log_2\frac{6}{10}\right)\right]=0.324\end{aligned}$$

特征“有自己的房子”的信息增益

$$\begin{aligned}g(D,\text{有自己的房子})&=H(D)-H(D|\text{有自己的房子})\\&=0.971-0.551=0.420\end{aligned}$$

特征“信贷情况”的信息增益

$$\begin{aligned}g(D,\text{信贷情况})&=H(D)-H(D|\text{信贷情况})\\&=0.971-0.608=0.363\end{aligned}$$

比较各特征的信息增益值，对于特征“有自己的房子”的信息增益值最大，所以选择特征“有自己的房子”作为最优特征，作为根结点的特征。

1.4 ID3 算法

ID3 算法最早是由罗斯昆(J. Ross Quinlan)1975 年提出的一种分类预测算法。ID3 算法的核心是在决策树各个子结点上应用**信息增益**准则选择特征，递归的构建决策树，具体方法是: 首先计算所有特征的信息增益，选择信息增益最大的特征作为根结点的特征，按照这一特征将训练数据集分割成子集，然后对于每个子集，分别计算其余的特征的信息增益，选择信息增益最大的特征作为子结点（建立子结点也称为**分裂**）；再对子结点递归调用以上方法，构建决策树。直到所有特征的信息增益均很小或没有特征可以选择为止，最后得到一个

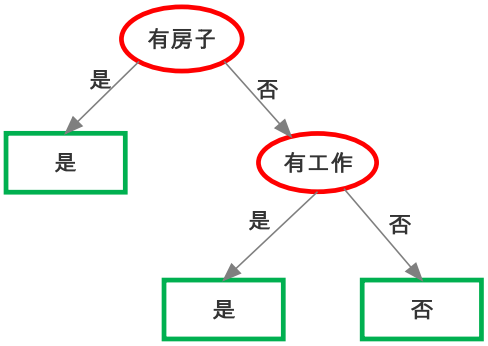
决策树。

继续前面的贷款例子。由于特征“有自己的房子”的信息增益值最大，所以选择“有自己的房子”作为根结点的特征。它将训练数据集划分为两个子集 D_1（A_3 取值为“是”）和 D_2（A_3 取值为“否”）。由于 D_1 只有同一类样本点，可以明确要贷款给 D_1，所以它成为一个叶结点，结点类标记为“是”。

对于 D_2 则需要从特征“年龄”，有工作和信贷情况中选择新的特征。计算各个特征的信息增益:

$$g(D_2, \text{年龄}) = H(D_2) - H(D_2 \mid \text{年龄}) = 0.918 - 0.667 = 0.251$$
$$g(D_2, \text{有工作}) = H(D_2) - H(D_2 \mid \text{有工作}) = 0.918$$
$$g(D_2, \text{信贷情况}) = H(D_2) - H(D_2 \mid \text{信贷情况}) = 0.474$$

选择信息增益最大的特征“有工作”作为结点的特征。特征有工作有 2 个取值，一个对应“是”(有工作)的子结点，包含 3 个样本，他们属于同一类，所以这是一个叶结点，类标记为“是”；另一个对应“否”(无工作)的子结点，包含 6 个样本，属于同一类，这也是一个叶结点，类标记为“否”。这样生成了如下图所示的决策树，该决策树只用了两个特征(有两个内部结点)。



ID3 算法只有树的生成，该算法生成的树容易产生过拟合，无实际应用价值，要加以改进。

1.5 C4.5 算法

用信息增益选择特征时偏向于选择取值多的特征，就是说在训练集中，某个特征所取的不同值的个数越多，那么越有可能拿它来作为分裂特征。例如一个训练集 D 中有 10 个样本，对于某一个特征 A，它分别取 1~10 这十个数（考虑取值最多的极端情形），如果对特征 A 进行分裂将会分成 10 个类 D_i，$i = 1,...,10$，那么对于每一个类 $H(D_i) = -1 \cdot \log_2 1 = 0$，从而 $H(D \mid A) = \sum_{i=1}^{10} p_i H(D_i) = 0$，特征 A 的信息增益 $g(D, A)$ 最大，但是很显然，以特征 A 作为分裂特征没有意义。正是基于此，C4.5 算法在 ID3 算法上进行了改进，采用了**信息增益率**(information gain ratio)选择特征。信息增益率是信息增益再除以特征的“**分裂信息熵**”(split information)，这样将信息增益规范化后能够抑制选择取值多的特征。

设特征 A 有 n 个不同的取值，根据特征 A 的取值将数据集 D 划分为 n 个子集 D_1，D_2，...，D_n，$|D_i|$ 为 D_i 的样本个数。训练数据集 D 关于特征 A 的**分裂信息熵** $H_A(D)$ 定义为

$$H_A(D) = -\sum_{i=1}^{n} \frac{|D_i|}{|D|} \log_2 \frac{|D_i|}{|D|}$$

特征 A 对训练数据集 D 的**信息增益率**定义为其信息增益与训练数据集 D 关于特征 A 的分裂信息熵 $H_A(D)$ 之比，即

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

C4.5 算法相对于 ID3 算法主要有以下几个改进：

- 用信息增益率来选择特征
- 在决策树的构造过程中对树进行剪枝
- 对连续型特征和不完整数据也能处理

对连续型特征的离散化方法如下：

1) 对特征的取值进行升序排序
2) 取两个特征取值之间的某个值作为阈值，一般取中点作为可能的划分点，将数据集分成两部分（大于阈值，小于阈值），计算每个可能的划分点的信息增益。
3) 选择信息增益最大的划分点作为该特征的最佳划分点
4) 计算最佳划分点的信息增益率作为特征的信息增益率。

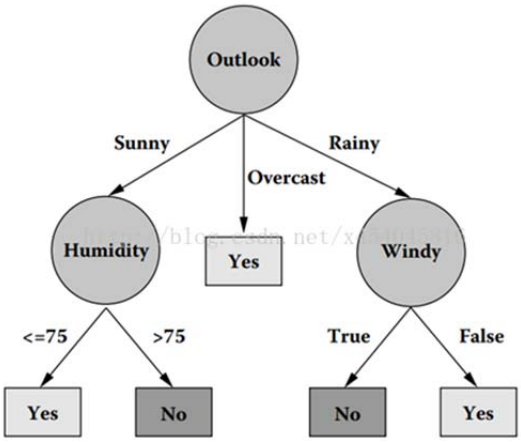
下面是包含连续型特征的例子。数据集如下表所示，它表示的是天气情况与去不去打高尔夫球之间的关系。

Day	Outlook	Temperature	Humidity	Windy	Play Golf?
1	Sunny	85	85	False	No
2	Sunny	80	90	True	No
3	Overcast	83	78	False	Yes
4	Rainy	70	96	False	Yes
5	Rainy	68	80	False	Yes
6	Rainy	65	70	True	No
7	Overcast	64	65	True	Yes
8	Sunny	72	95	False	No
9	Sunny	69	70	False	Yes
10	Rainy	75	80	False	Yes
11	Sunny	75	70	True	Yes
12	Overcast	72	90	True	Yes
13	Overcast	81	75	False	Yes
14	Rainy	71	80	True	No

对于连续型特征，取两个特征值的中点作为可能的划分点，计算每个划分点的信息增益。需要计算 $N-1$ 次（ N 个样本），计算量是相当大的。为了减少计算量，对特征值先进行从小到大排序，只有在类别发生改变的地方才进行分裂。比如对 Temperature 进行排序如下：

Outlook	Temperature	Humidity	Windy	PlayGolf?
overcast	64	65	TRUE	yes
rainy	65	70	TRUE	no
rainy	68	80	FALSE	yes
sunny	69	70	FALSE	yes
rainy	70	96	FALSE	yes
rainy	71	91	TRUE	no
sunny	72	95	FALSE	no
overcast	72	90	TRUE	yes
rainy	75	80	FALSE	yes
sunny	75	70	TRUE	yes
sunny	80	90	TRUE	no
overcast	81	75	FALSE	yes
overcast	83	86	FALSE	yes
sunny	85	85	FALSE	no

14 个样本有 13 个可能的划分点，如果只有在类别发生改变的时候才进行分裂，只需计算 7 个。选择信息增益最大的划分点作为特征“Temperature”的最优划分点。对连续型特征“Humidity”可以类似处理。以信息增益率代替信息增益作为特征选择方法，类似 ID3 算法， 运用 C4.5 算法对数据集产生如下的决策树。



C4.5 算法有如下优点：产生的分类规则易于理解，准确率较高。其缺点是：在构造树的过程中，需要对数据集进行多次的顺序扫描和排序，因而导致算法的低效。此外，C4.5 只适合于能够驻留于内存的数据集，当训练集大得无法在内存容纳时程序无法运行。

1.6 CART 算法

分类与回归树 （Classification And Regression Tree, CART）其核心思想与 ID3 和 C4.5 相同，同样由特征选取、树的生成和剪枝组成，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和否。决策树递归地二分每个特征，将输入空间**划分**为有限个单元。

CART 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划分点。

分类问题中，假设有 K 个类别，样本点属于第 k 类的概率为 p_k ，则其概率分布 P 的**基尼指数**（gini index）定义为

$$Gini(P)=\sum_{k=1}^{K}p_k(1-p_k)=1-\sum_{k=1}^{K}p_k^2$$

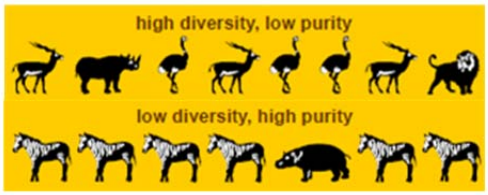
特别地，对于二类分类问题，如样本点属于第一个类的概率是 p ，则其概率分布 P 的基尼指数为

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

对于给定样本集合 D ，其基尼指数为

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

其中， C_k 是集合中属于第 k 类的样本子集。 $Gini(D)$ 又称为集合 D 的**基尼不纯度**(gini imputity)，从定义来看，其含义是“任意取两个样本其属于不同类的概率”。因此 $Gini(D)$ 越小，则集合 D 的“纯度”越高，即集合 D 的样本越接近属于同一类。基尼指数和生态学中的辛普森多样性指数（Simpson's Diversity Index）含义相同。如下图，基尼指数等于不同物种相遇的概率。



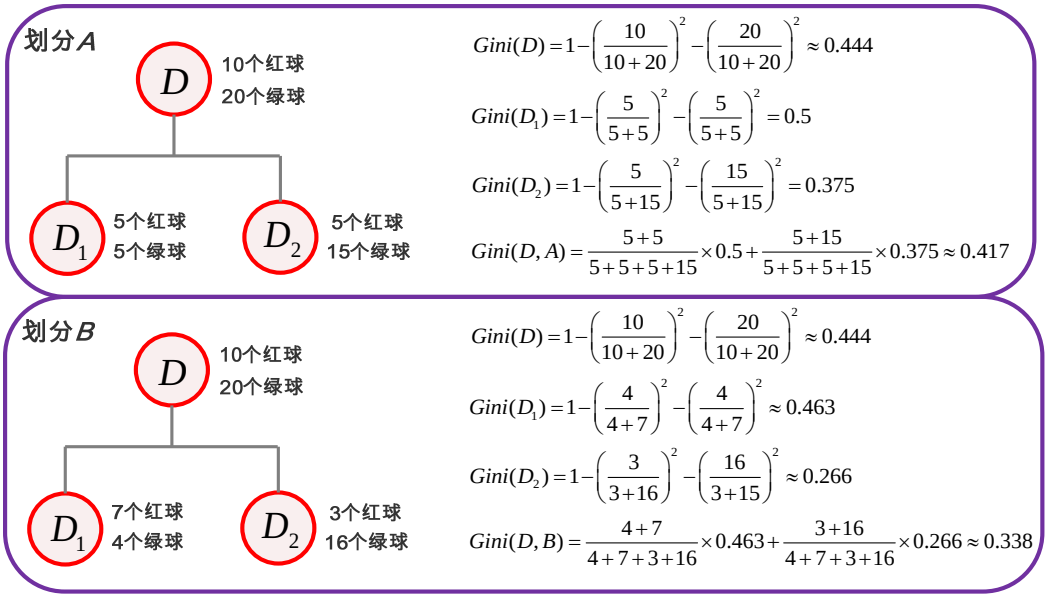
$$Gini=1-2(3/8)^2-2(1/8)^2=0.69$$

$$Gini=1-(6/7)^2-2(1/7)^2=0.24$$

如果样本集合 D 根据特征 A 是否取某一可能值 a 被划分成 D_1 和 D_2 两部分，则在特征 A 的条件下，**集合 D 的基尼指数**为

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

基尼指数 $Gini(D)$ 表示集合 D 的分类不纯度，$Gini(D,A)$ 表示经过特征 A 划分后集合 D 的分类不纯度。基尼指数值越大，样本集合的分类不确定性也就越大，这一点和熵类似。



上图显示了在不同划分下，如何计算划分后的基尼指数。

类似信息增益，我们也可以计算基于特征 A 对集合 D 进行划分的前后，集合 D 基尼指数的改变，即**基尼指数增益**：

$$\Delta Gini(A) = Gini(D) - Gini(D, A)$$

$\Delta Gini(A)$ 描述的是划分前后集合 D 的不纯度的变化：基尼不纯度改进越大，说明分裂后各个子结点任取两个样本不同类的概率越低。**在决策树选择特征时，应选择基尼指数增益值最大的特征，或等价地，选择基尼指数最小的特征，作为该节点分裂的最优特征。**

下面介绍基于基尼指数的分类树的生成过程。根据训练数据集，从根节点开始，递归地对每个结点进行以下操作，构建二叉决策树：

1. 设结点的训练数据集为 D ，计算现有特征对该数据集的基尼指数，此时，对每一个特征 A ，对其可能取的每个值 a ，根据样本点对 $A = a$ 的测试为“是”或“否”将 D 划分成 D_1 和 D_2 两部分，计算 $A = a$ 的基尼指数 $Gini(D, A)$ 。
2. 在所有可能的特征 A 以及它们所有可能的划分点 a 中，选择基尼指数最小的特征及其对应的划分点作为最优特征与最优划分点。依最优特征与最优划分点，从现结点生成两个子结点，将训练数据集依特征分配到两个子节点中去。
3. 对两个子节点递归地调用步骤（1）、（2），直至满足停止条件。
4. 生成 CART 决策树。

算法停止的条件是节点中的样本个数小于预定的阈值，或样本集的基尼指数小于预定阈值，或者没有更多特征。

下面继续考虑前面的贷款例子，采用其数据集构造 CART 分类决策树。

特征“年龄”的基尼指数为

$$Gini(D, \text{年龄=青年}) = \frac{5}{15}\left(2 \times \frac{2}{5} \times (1 - \frac{2}{5})\right) + \frac{10}{15}\left(2 \times \frac{7}{10} \times (1 - \frac{7}{10})\right) = 0.44$$

类似可以计算:

$$Gini(D, \text{年龄=中年}) = 0.48$$

$$Gini(D, \text{年龄=老年}) = 0.44$$

由于 $Gini(D, \text{年龄=青年})$ 和 $Gini(D, \text{年龄=老年})$ 相等，且最小，所以，青年和老年都可以作为年龄的划分点。

特征“有工作”和“有自己的房子”的基尼指数分别为：

$$Gini(D, \text{有工作=是}) = 0.32$$

$$Gini(D, \text{有自己的房子=是}) = 0.27$$

由于特征“有工作”和“有自己的房子”只有一个划分点，所以它们就是最优划分点。

特征“信贷情况”的基尼指数为：

$$Gini(D, \text{信贷情况=非常好}) = 0.36$$

$$Gini(D, \text{信贷情况=好}) = 0.47$$

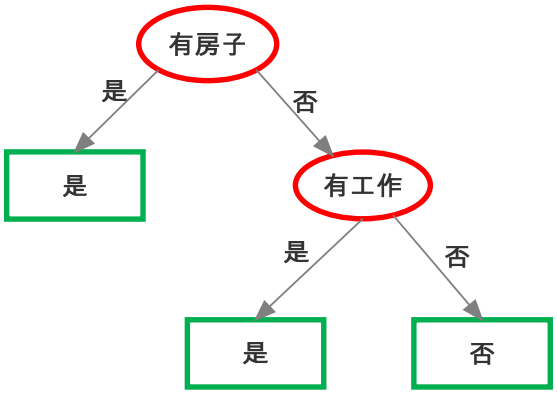
$$Gini(D, \text{信贷情况=一般}) = 0.32$$

$Gini(D, \text{信贷情况=一般})$ 最小，所以“信贷情况=一般”为特征“信贷情况”的最优划分点。

在 4 个特征：年龄、有工作、有自己的房子、信贷情况中，

$$Gini(D,\text{有自己的房子=是}) = 0.27$$

最小，所以选择特征“有自己的房子”作为最优特征，“有自己的房子=是”为其最优划分点。于是根节点“有自己的房子”生成两个结点，一个都是同一类为叶结点，对另一个结点继续使用以上的方法在另外的 3 个特征中选择最优特征及其最优划分点，结果是“有工作=是”。依此再次分裂，所得结点都是叶结点。对于本问题，按照 CART 算法说生成的决策树与按照 ID3 算法所生成的决策树完全一致，即得到如下的二叉分类决策树。



CART 回归树的生成

回归树用平方误差最小化准则进行特征选择生成二叉树。假设 X 与 Y 分别为输入和输出变量，且 Y 是连续变量，给定训练集

$$D = \{(x_1, y_1), (x_2, y_2), ..., (x_m, y_m)\}$$

一个回归树对应着输入空间（特征空间）的一个划分以及在划分单元上的输出值。假设已经将输入空间划分成 K 个区域 $R_1, R_2, ..., R_K$，并且在每个区域 R_k 上有一个固定的输出值 c_k，于是回归树模型可以表示为：

$$f(x) = \sum_{k=1}^{K} c_k I(\boldsymbol{x} \epsilon R_k)$$

其中 I 为指示函数，$I(true) = 1$，$I(false) = 0$。

当用平方误差 $\sum_{x_i \epsilon C_k} (y_i - f(x_i))^2$ 来表示回归树对训练数据的预测误差时候，选取平方误差最小时候的特征用来作为构建二叉树的分裂特征。问题是怎样对输入空间进行划分。这里采用启发式的方法，选择第 j 个特征 $x^{(j)}$ 和它的取值 s 作为分裂特征和划分点，将数据集分成两个区域：

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

对于两个区域的输出值，用该区域中 y 的平均值来代替:

$$c_1 = ave\{y_i \mid x_i \epsilon R_1(j,s)\}$$
$$c_2 = ave\{y_i \mid x_i \epsilon R_2(j,s)\}$$

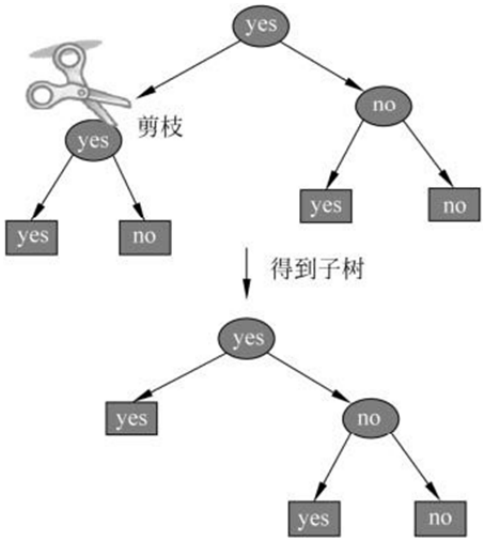
选取最小平方误差时候的 j ， s 得到最优分裂特征和最优划分点。具体地，求解

$$\min_{j,s}\left\{\min_{c_1}\sum_{x_i\epsilon R_1(j,s)}(y_i-c_1)^2+\min_{c_2}\sum_{x_i\epsilon R_2(j,s)}(y_i-c_2)^2\right\}$$

对大括号内的计算是对第 j 个特征，找出其最小平方误差的划分，其中两个最小值，返回的就是左右子结点划分的平方误差。

1.7 决策树的剪枝

剪枝目的是避免决策树过拟合。前面的算法生成的决策树非常详细并且庞大，每个特征都被详细地加以考虑，决策树的叶结点所覆盖的训练样本都是“纯”的（即同一类）。训练样本中的噪声数据也会被决策树学习，成为决策树的部分，因此用这个决策树来对训练样本进行分类的话，你会发现对于训练样本而言，这个树表现完好，误差率极低且能够正确地得对训练集中的样本进行分类，但是对于测试数据的表现就效果不好。我们需要对生成的决策树进行剪枝。



剪枝方法主要有两类：先剪枝和后剪枝

- **先剪枝**（pre-pruning）：在建树的过程中，当满足一定条件，例如，信息增益或者某个统计量达到某个预先设定的阈值时，结点不再继续分裂，内部结点成为一个叶结点。叶结点取子集中频率最大的类作为自己的标识，或者可能仅仅存储这些实例的概率分布函数。
- **后剪枝**（Pos-pruning）：它是由“完全生长”的树剪去分支，首先生成与训练数据集合完全拟合的一棵决策树，然后从树的叶子开始剪枝，逐步向根的方向剪。剪枝时要用到一个**测试数据集合**（Adjusting set），如果存在某个叶子剪去后能使得在测试集上的准确度不降低，于是剪去该叶子，最终形成一棵结点数尽可能少的决策树。

1.8 决策树算法小结

下表给出了决策树算法 ID3，C4.5 和 CART 的一个比较总结。

算法	支持模型	树结构	特征选择	连续值处理	缺失值处理	剪枝
ID3	分类	多叉树	信息增益	不支持	不支持	不支持
C4.5	分类	多叉树	信息增益率	支持	支持	支持
CART	分类，回归	二叉树	基尼指数，均方差	支持	支持	支持

我们来看看决策树算法的优缺点。

- 决策树算法的优点：

1）生成的决策树很直观，相比于神经网络之类的黑盒分类模型，决策树在逻辑上可以得到很好的解释。

2）数据基本不需要预处理，不需要提前归一化，处理缺失值。

3）既可以处理离散型特征值也可以处理连续型特征值。

4） 对于异常点的容错能力好，健壮性高。

- 决策树算法的缺点:

1）决策树算法非常容易过拟合，导致泛化能力不强。可以通过设置节点最少样本数量和限制决策树深度来改进。

2）决策树会因为样本发生一点点的改动，就会导致树结构的剧烈改变。这个可以通过集成学习（随机森林）之类的方法解决。

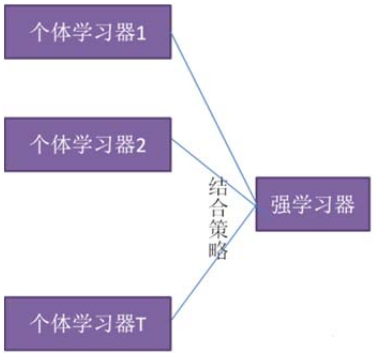
3）有些比较复杂的关系，决策树很难学习，比如异或。一般这种关系可以换神经网络分类方法来解决。

2 集成学习

集成学习(ensemble learning)可以说是现在非常火爆的机器学习方法了。它本身不是一个单独的机器学习算法，而是通过构建并结合多个机器学习器来完成学习任务。也就是我们常说的“博采众长”。集成学习可以用于分类问题集成，回归问题集成，特征选取集成，异常点检测集成等等，可以说所有的机器学习领域都可以看到集成学习的身影。

2.1 集成学习概述

从下图，我们可以对集成学习的思想做一个概括。对于训练集数据，我们通过训练若干个个体学习器，通过一定的结合策略，就可以最终形成一个强学习器，以达到博采众长的目的。



也就是说，集成学习有两个主要的问题需要解决，第一是如何得到若干个个体学习器，第二是如何选择一种结合策略，将这些个体学习器集合成一个强学习器。

2.2 集成学习之个体学习器

前面我们讲到，集成学习的第一个问题就是如何得到若干个个体学习器。这里我们有两种选择。

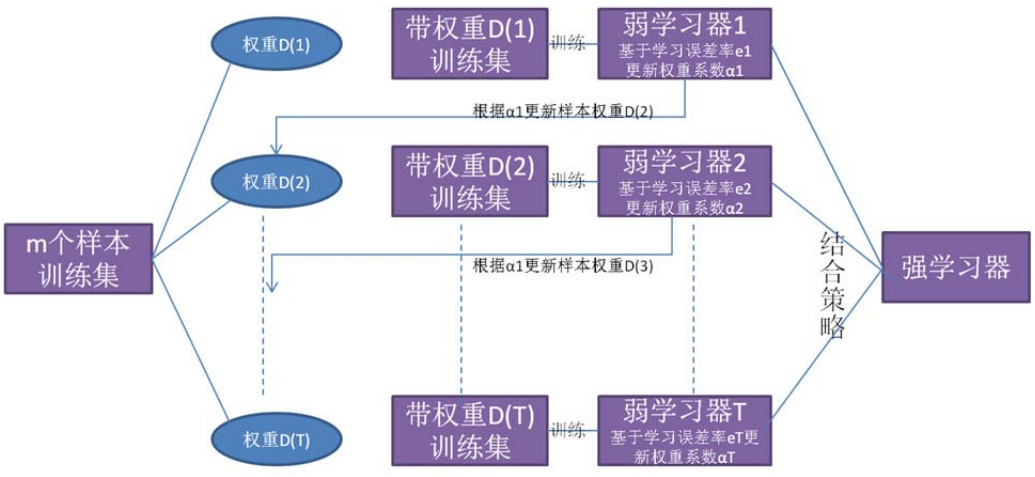
第一种就是所有的个体学习器都是一个种类的，或者说是同质的。比如都是决策树个体学习器，或者都是神经网络个体学习器。第二种是所有的个体学习器不全是一个种类的，或者说是异质的。比如我们有一个分类问题，对训练集采用支持向量机个体学习器，逻辑回归个体学习器和朴素贝叶斯个体学习器来学习，再通过某种结合策略来确定最终的分类强学习器。

目前来说，同质个体学习器的应用是最广泛的，一般我们常说的集成学习的方法都是指的同质个体学习器。而同质个体学习器使用最多的模型是 CART 决策树和神经网络。同质个体学习器按照个体学习器之间是否存在依赖关系可以分为两类，第一个是个体学习器之间存在强依赖关系，一系列个体学习器基本都需要串行生成，代表算法是 boosting 系列算法，第二个是个体学习器之间不存在强依赖关系，一系列个体学习器可以并行生成，代表算法是 bagging 和随机森林系列算法。下面就分别对这两类算法做一个概括总结。

2.3 集成学习之 Boosting

对于分类问题而言，给定一个训练数据，求一个比较粗糙的分类器（即弱分类器）要比求一个精确的分类器（即强分类器）容易得多。提升（Boosting）方法就是从弱学习算法出发，反复学习，得到一系列弱分类器，然后组合这些弱分类器，构成一个强分类器。大多数的提升方法都是改变训练数据的概率分布（训练数据中的各个数据点的权值分布），调用弱学习算法得到一个弱分类器，再改变训练数据的概率分布，再调用弱学习算法得到一个弱分类器，如此反复，得到一系列弱分类器。

Boosting 的算法原理我们可以用一张图做一个概括如下：



从图中可以看出，Boosting 算法的工作机制是首先从训练集用初始权重训练出一个弱学习器 1，根据弱学习器 1 的**学习误差率表现来更新训练样本的权重**，使得之前弱学习器 1 学习误差率高的训练样本点的权重变高，使得这些误差率高的点在后面的弱学习器 2 中得到更多的重视。然后基本调整权重后的训练集来训练弱学习器 2。如此重复进行，直到弱学习器数达到事先指定的数目 T，最终将这 T 个弱学习器通过集合策略进行整合，得到最终的强学习器。

对于提升方法来说，有两个问题需要回答：一是在每一轮如何改变训练数据的概率分布；二是如何将多个弱分类器组合成一个强分类器。

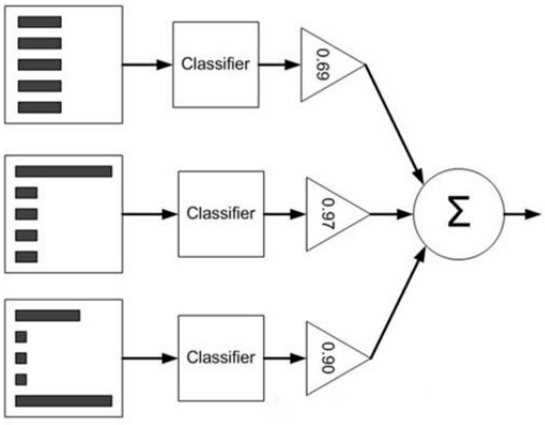
AdaBoost（Adaptive Boosting）是指自适应提升算法，能够自适应地改变训练样本的概率分布，使得基分类器聚焦在那些很难分的样本上。它根据每次训练集之中每个样本的分类是否正确，以及上次分类的准确率，来确定每个样本的权值，降低分类正确的样本权重，提高分类错误的样本的权重。将修改过权值的新数据集送给下层分类器进行训练。至于第二个问题，AdaBoost 采取加权多数表决的方法。具体地，加大分类误差率小的弱分类器的权值，使其在表决中起较大的作用，减小分类误差率大的弱分类器的权值，使其在表决中起较小的作用。整个过程如下所示：

1. 先通过对 m 个训练样本的学习得到第一个弱分类器，每个样本的权重被初始化为相等的值，并计算弱分类器的错误率；

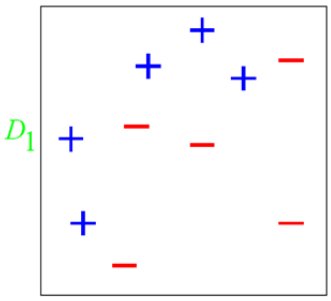
2. 将分错的样本和其他的新数据一起构成一个新的 m 个的训练样本，通过对这个样本的学习得到第二个弱分类器，其中分类正确的样本权重降低，分类错误的样本权重提高；

3. 将 1 和 2 都分错了的样本加上其他的新样本构成另一个新的 m 个的训练样本，通过对这个样本的学习得到第三个弱分类器；

4. 最终经过提升的强分类器。即某个数据被分为哪一类要由各弱分类器权值决定，各弱分类器的权重基于其分类错误率计算。

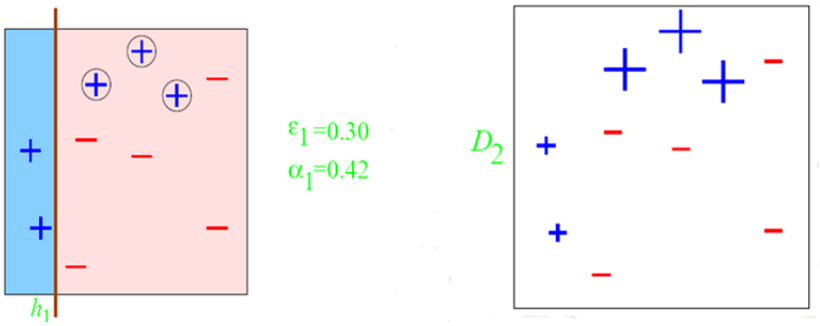


下面我们举一个简单的例子来看看 AdaBoost 的实现过程：

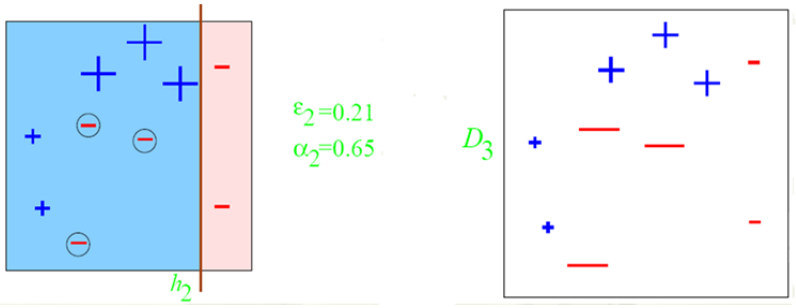


图中，“+”和“-”分别表示两种类别，在这个过程中，我们使用水平或者垂直的直线作为分类器，来进行分类。

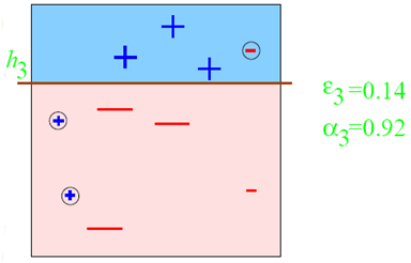
- 第一步：根据分类的正确率，得到一个新的样本分布 D_2 ，一个子分类器 h_1 ，其中划圈的样本表示被分错的。在右边的图中，比较大的“+”表示对该样本权重提高。



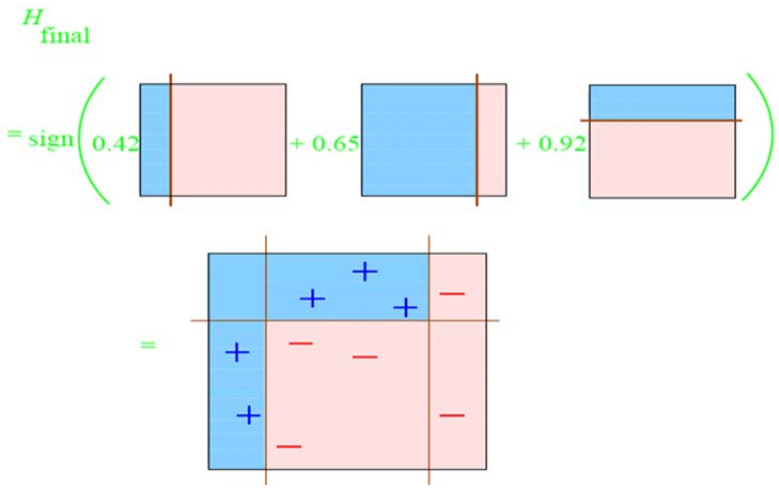
- 第二步： 根据分类的正确率，得到一个新的样本分布 D_3 ，一个子分类器 h_2



- 第三步： 得到一个子分类器 h_3



- 第四步：整合所有子分类器

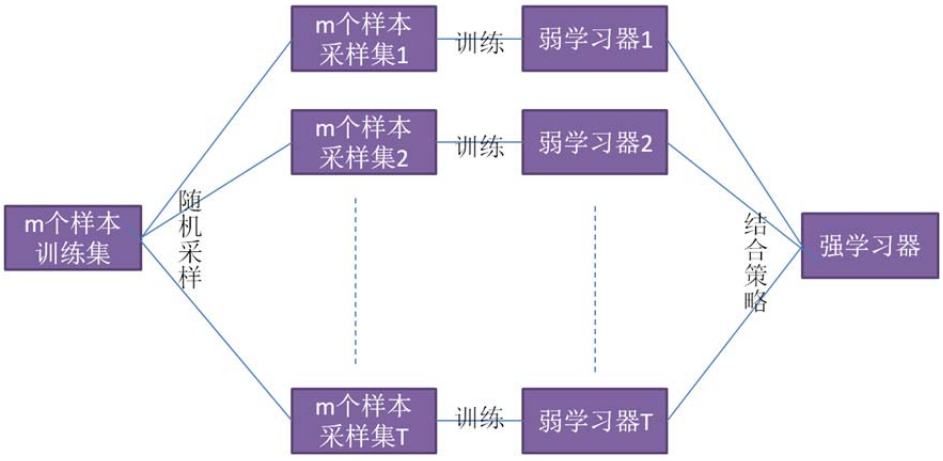


从结果中看，即使简单的分类器，组合起来也能获得很好的分类效果。

对 AdaBoost 算法的研究以及应用大多集中于分类问题，同时也出现了一些在回归问题上的应用。Adaboost 可以使用各种方法构建弱分类器，如使用 SVM，或决策树等，最后得到的强分类器通常具有非常好的效果，而且很少过拟合。Boosting 算法中除了 AdaBoost ,应用比较广泛的还有**梯度提升树**(Gradient Boosting Tree) ,后面将专门介绍。

2. 4 集成学习之 Bagging

Bagging 是 Bootstrap AGGregatING (自助聚集)的缩写。Bagging 的算法原理和 Boosting 不同，它的弱学习器之间没有依赖关系，可以并行生成，如下图所示：



从图中看出，Bagging 的个体弱学习器的训练集是通过**随机采样**得到的。通过 T 次的随机采样，我们就可以得到 T 个采样集，对于这 T 个采样集，我们可以分别独立的训练出 T 个弱学习器，再对这 T 个弱学习器通过集合策略来得到最终的强学习器。

对于这里的随机采样有必要做进一步的介绍，这里一般采用的是**自助采样法（Bootstap）**,即对于 m 个样本的原始训练集 ,我们每次先随机采集一个样本放入采样集，接着把该样本放回，也就是说下次采样时该样本仍有可能被采集到，这样采集 m 次，最终

可以得到 m 个样本的采样集，由于是随机采样，这样每次的采样集是和原始训练集不同的（**注意可能有样本是重复的**），和其他采样集也是不同的，这样得到多个不同的弱学习器。

Bootstrap 本义是指高靴子口后面的悬挂物、小环、带子，是穿靴子时用手向上拉的工具。“pull up by your own bootstraps”即“通过拉靴子让自己上升”，意思是“不可能发生的事情”。后来意思发生了转变，隐喻“不需要外界帮助，仅依靠自身力量让自己变得更好”。

由多个决策树构成的随机森林基于 Bagging，样本随机有放回采样（Bootstap）构成每个子树的数据集。随机森林算法的原理在后面介绍。

2.5 集成学习之结合策略

在上面几节里面我们主要关注于学习器，提到了学习器的结合策略但没有细讲，本节就对集成学习之结合策略做一个总结。我们假定我得到的T个弱学习器是$\{h_1,h_2,...h_T\}$

- **平均法**

对于数值类的回归预测问题，通常使用的结合策略是平均法，也就是说，对于若干弱学习器的输出进行平均得到最终的预测输出。

最简单的平均是算术平均，也就是说最终预测是

$$H(x)=\frac{1}{T}\sum_{1}^{T}h_i(x)$$

如果每个弱学习器有一个权重w，则最终预测是

$$H(x)=\frac{1}{T}\sum_{i=1}^{T}w_ih_i(x)$$

其中w_i是弱学习器h_i的权重，通常有$w_i\geq 0,\ \sum_{i=1}^{T}w_i=1$。

- **投票法**

对于分类问题的预测，我们通常使用的是投票法。假设我们的预测类别是$\{c_1,c_2,...c_K\}$，对于任意一个预测样本$\boldsymbol{x}$，我们的T个弱学习器的预测结果分别是$(h_1(\boldsymbol{x}),h_2(\boldsymbol{x})...h_T(\boldsymbol{x}))$。

最简单的投票法是相对多数投票法，也就是我们常说的少数服从多数，也就是T个弱学习器的对样本$\boldsymbol{x}$的预测结果中，数量最多的类别为最终的分类类别。

稍微复杂的投票法是绝对多数投票法，也就是我们常说的要票过半数。在相对多数投票法的基础上，不光要求获得最高票，还要求票过半数。否则会拒绝预测。

更加复杂的是加权投票法，和加权平均法一样，每个弱学习器的分类票数要乘以一个权重，最终将各个类别的加权票数求和，最大的值对应的类别为最终类别。

- **学习法**

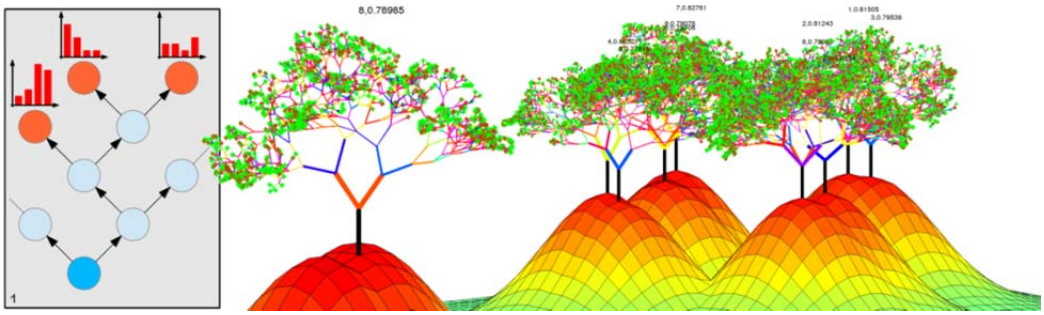
前面的方法都是对弱学习器的结果做平均或者投票，相对比较简单，但是可能学习误差较大，于是就有了学习法这种方法，对于学习法，代表方法是 **stacking**，当使用 stacking 的结合策略时， 我们不是对弱学习器的结果做简单的逻辑处理，而是再加上一层学习器，也就是说，我们将训练集弱学习器的学习结果作为输入，将训练集的输出作为输出，重新训练一个学习器来得到最终结果。

在这种情况下，我们将弱学习器称为初级学习器，将用于结合的学习器称为次级学习器。对于测试集，我们首先用初级学习器预测一次，得到次级学习器的输入样本，再用次级学习器预测一次，得到最终的预测结果。

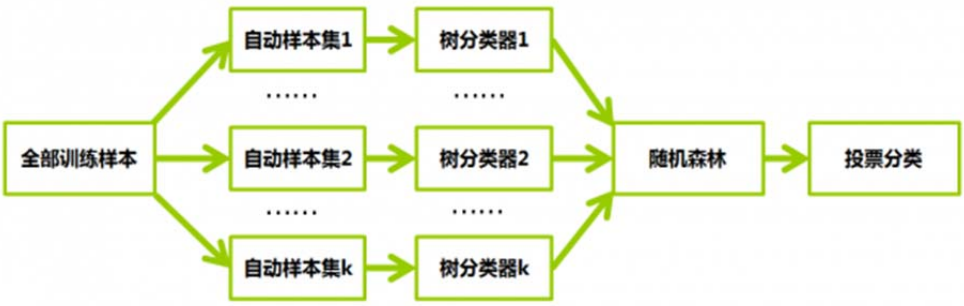
3 随机森林

3.1 什么是随机森林？

作为新兴起的、高度灵活的一种机器学习算法，**随机森林**（Random Forest，RF）拥有广泛的应用前景，从市场营销到医疗保健保险，既可以用来做市场营销模拟的建模，统计客户来源，保留和流失，也可用来预测疾病的风险和病患者的易感性。



随机森林中有许多的分类树。我们要将一个输入样本进行分类，我们需要将输入样本输入到每棵树中进行分类。打个比喻：森林中召开会议，讨论某个动物到底是老鼠还是松鼠，每棵树都要独立地发表自己对这个问题的看法，也就是每棵树都要投票。该动物到底是老鼠还是松鼠，要依据投票情况来确定，获得票数最多的类别就是森林的分类结果。将若干个弱分类器的分类结果进行投票选择，从而组成一个强分类器，这就是随机森林 bagging 的思想。



随机森林是一种集成学习+决策树的分类模型，它可以利用集成的思想（投票选择的策略）来提升单颗决策树的分类性能（通俗来讲就是“三个臭皮匠，顶一个诸葛亮”）。

集集成学习和决策树于一身，随机森林算法具有众多的优点，其中最为重要的就是在随机森林算法中每棵树都尽最大程度的生长，并且没有剪枝过程。

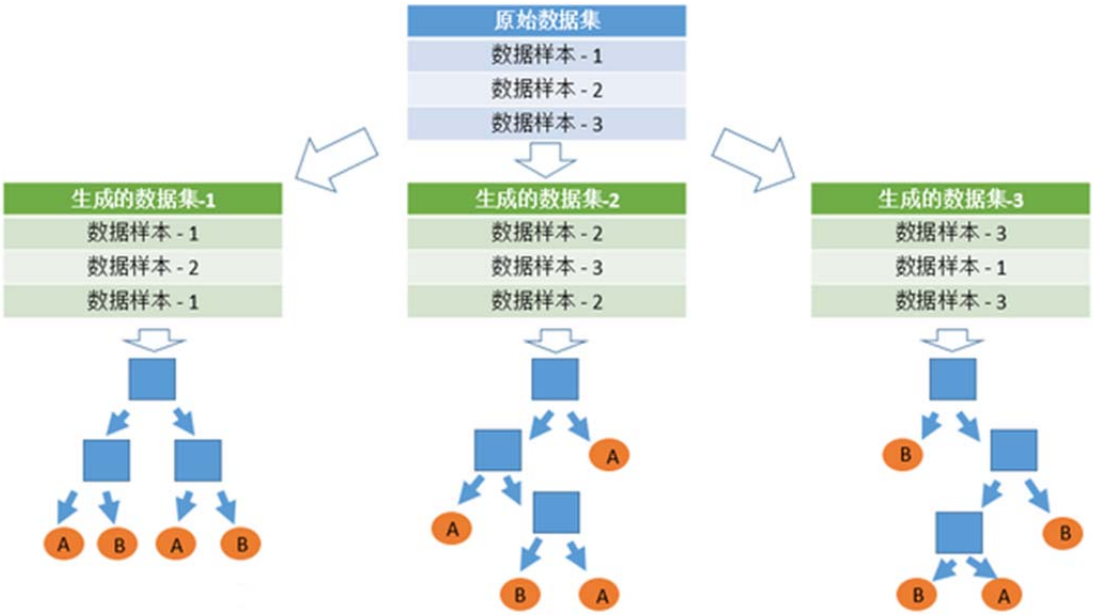
随机森林引入了两个随机性——**随机选择样本**（bootstrap sample）和**随机选择特征**进行训练。两个随机性的引入对随机森林的分类性能至关重要。由于它们的引入，使得随机森林不容易陷入过拟合，并且具有很好得抗噪能力（比如：对缺省值不敏感）。

3.2 随机森林的构建

那随机森林具体如何构建呢？有两个方面：数据的随机性选取，以及待选特征的随机选取。

- **数据的随机选取**

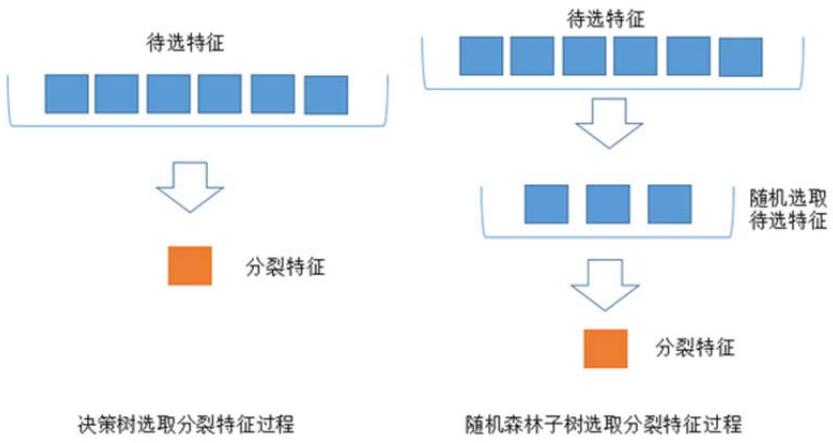
首先，从原始的数据集中采取**有放回的抽样**，构造子数据集，子数据集的数据量是和原始数据集相同的。不同子数据集的元素可以重复，同一个子数据集中的元素也可以重复。然后，利用子数据集来构建子决策树，将这个数据放到每个子决策树中，每个子决策树输出一个结果。最后，如果有了新的数据需要通过随机森林得到分类结果，就可以通过对子决策树的判断结果的**投票**，得到随机森林的输出结果了。如下图，假设随机森林中有 3 棵子决策树，2 棵子树的给出的分类结果是 A 类，1 棵子树的给出的分类结果是 B 类，那么随机森林给出的分类结果就是 A 类。注意到从原始数据集随机有放回抽样生成的子决策树的数据集，是有重复样本的。



- **待选特征的随机选取**

与数据集的随机选取类似，随机森林中的子树的每一个分裂过程并未用到所有的待选特征，而是从所有的待选特征中随机选取一定的特征，之后再在随机选取的特征中选取最优的特征。这样能够使得随机森林中的决策树都能够彼此不同，提升系统的多样性，从而提升分类性能。

下图中，蓝色的方块代表所有可以被选择的特征，也就是目前的待选特征。黄色的方块是分裂特征。左边是一棵决策树的特征选取过程，通过在待选特征中选取最优的分裂特征，完成分裂。右边是一个随机森林中的子树的特征选取过程。



有了树我们就可以分类了，但是森林中的每棵树是怎么生成的呢？

每棵树的按照如下规则生成：

1）如果训练集大小为 m ，对于每棵树而言，随机且有放回地从训练集中的抽取 m 个训练样本，作为该树的训练集；

从这里我们可以知道：每棵树的训练集都是不同的，而且里面包含重复的训练样本。

2）如果每个样本的特征维度为 n ，指定一个常数 $N << n$ ，随机地从 n 个特征中选取 N 个特征子集，每次树进行分裂时，从这 N 个特征中选择最优的；

3）每棵树都尽最大程度的生长，并且没有剪枝过程。

随机森林分类效果（错误率）与两个因素有关：

- 森林中任意两棵树的相关性：相关性越大，错误率越大；
- 森林中每棵树的分类能力：每棵树的分类能力越强，整个森林的错误率越低。

减小特征选择个数 N ，树的相关性和分类能力也会相应的降低；增大 N ，两者也会随之增大。所以关键问题是如何选择最优的 N （或者是范围），这也是随机森林唯一的一个参数。

构建随机森林的关键问题就是如何选择最优的 N ，要解决这个问题主要依据计算**袋外错误率** oob error（out-of-bag error）。

随机森林有一个重要的优点就是，没有必要对它进行交叉验证或者用一个独立的测试集来获得误差的一个无偏估计。它可以在内部进行评估，也就是说在生成的过程中就可以对误差建立一个无偏估计。

在构建每棵树时，我们对训练集使用了不同的 bootstrap sample（随机且有放回地抽取）。所以对于每棵树而言（假设对于第 k 棵树），大约有 1/3 的训练实例没有参与第 k 棵树的生成，它们称为第 k 棵树的 oob 样本。

而这样的采样特点就允许我们进行 oob 估计，它的计算方式如下：

1）对每个样本，计算它作为 oob 样本的树对它的分类情况（约 1/3 的树）；

2）然后以简单多数投票作为该样本的分类结果；

3）最后用误分个数占样本总数的比率作为随机森林的 oob 错误率。

4 梯度提升树（GBDT）

4.1 梯度提升（Gradient Boosting）

梯度提升（Gradient Boosting）是通过组合多个弱学习器构成一个强学习器的提升算法，弱学习器也称为**基学习器**，通常采用决策树。类似其它提升算法，梯度提升也是基于迭代方式产生新的基学习器。假设在第 t 轮迭代得到不是很完美的模型 f_t，梯度提升算法不是去改变 f_t，而是通过加上一个基学习器 h 构建一个更好的模型 $f_{t+1}(x)=f_t(x)+h(x)$。问题是如何找到 h？注意到完美的 h 满足

$$f_{t+1}(x)=f_t(x)+h(x)=y$$

其中 y 为真实值。或等价地

$$h(x)=y-f_t(x)$$

因此，h 需要拟合**残差** $y-f_t(x)$。这样的 h 适合回归问题。其算法思想可以用一个通俗的例子解释，假如有个人 30 岁，我们首先用 20 岁去拟合，发现损失有 10 岁，这时我们用 6 岁去拟合剩下的损失，发现差距还有 4 岁，第三轮我们用 3 岁拟合剩下的差距，差距就只有一岁了。如果我们的迭代轮数还没有完，可以继续迭代下面，每一轮迭代，拟合的岁数误差都会减小。

注意到 $y-f(x)$ 是误差平方损失函数 $\frac{1}{2}(y-f(x))^2$ 关于 f 的负梯度（这里梯度意思是偏导数，而非向量）。我们把误差平方损失函数推广到其它可微损失函数，以便适应不同的分类问题。在生成每个新的基学习器的时候采用梯度下降的思想，并在使损失函数变小的**负梯度方向**上建立一个新的基学习器，这样就达到损失函数不断减小的目的。

给定训练集 $\{(x_1,y_1),\cdots,(x_m,y_m)\}$，其中输入变量 $\boldsymbol{x}$ 为特征，输出变量 y 为相应的类别标签（分类问题）或实值（回归问题）。对于分类问题或回归问题，给定相应的损失函数 $L(y,f(x))$，其中 f 是描述输入变量 $\boldsymbol{x}$ 到输出变量 y 的真实映射。对于一般学习问题，学习算法的目的是找到使损失函数在训练集上的期望值最小的函数 $f(x)$ 的近似 $\hat{f}(x)$：

$$\hat{f}(x)=\arg\min_f \mathrm{E}_{x,y}\left[L(y_i,f(x)\right]$$

梯度提升算法假定 y 是实值，把 f 近似为一系列基(或弱)学习器 $h_i(x)$ 的加权组合，$h_i(x)\in \mathrm{H}$，H 为某个模型类：

$$f(x)=\sum_{i=1}^{M}\gamma_i h_i(x)+const.$$

根据**经验风险最小化**原则，寻求使训练集上的损失函数的总和（或平均值）最小的 f 的近似 $\hat{f}$。首先从一个常数函数 $f_0(x)$ 开始，然后以贪心策略不断迭代得到其近似：

$$f_0(x)=\arg\min_{\gamma}\sum_{i=1}^{m}L(y_i,\gamma)$$

$$f_t(x)=f_{t-1}(x)+\arg\min_{h\in \mathrm{H}}\sum_{i=1}^{m}L(y_i,f_{t-1}(x_i)+h(x_i))$$

其中 h 是 H 中的某个基学习器。

上式右边的最小化问题通常难以求解。采用梯度下降的思想，上面的迭代公式修改为

$$f_t(x)=f_{t-1}(x)-\gamma_t\sum_{i=1}^{m}\nabla_{f_{t-1}}L(y_i,f_{t-1}(x_i))$$

$$\gamma_t = \arg\min_{\gamma} \sum_{i=1}^{m} L\left(y_i, f_{t-1}(x_i) - \gamma \frac{\partial L(y_i, f_{t-1}(x_i))}{\partial f_{t-1}(x_i)} \right)$$

当损失函数为误差平方时，即 $L(y, f(x)) = \frac{1}{2}(y - f(x))^2$ ，L 关于 f 的负梯度为 $y - f(x)$ 即为残差。对于一般的损失函数的负梯度 $-\frac{\partial L(y, f(x))}{\partial f(x)}$ ，称之为**伪残差**。

综上，梯度提升算法如下：

输入：训练集 $\{(x_i, y_i)\}_{i=1}^{m}$，可微损失函数 $L(y, f(x))$，迭代次数。

1. 初始化模型为常数值：

$$f_0(x) = \arg\min_{\gamma} \sum_{i=1}^{m} L(y_i, \gamma)$$

2. 迭代生成 M 个基学习器

a) 计算伪残差 $r_{ti} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)} \right]_{f(x)=f_{t-1}(x)}$ ， $i = 1, ..., m$

b) 基于 $\{(x_i, r_{ti})\}_{i=1}^{m}$ 生成基学习器 $h_t(x)$

c) 计算最优的 γ_t

$$\gamma_t = \arg\min_{\gamma} \sum_{i=1}^{m} L\left(y_i, f_{t-1}(x_i) - \gamma \frac{\partial L(y_i, f_{t-1}(x_i))}{\partial f_{t-1}(x_i)} \right)$$

d) 更新模型

$$f_t(x) = f_{t-1}(x) + \gamma_t h_t(x)$$

3. 输出 $f_M(x)$。

4.2 梯度提升树 GBDT

梯度提升算法中最典型的基学习器是决策树，尤其是 CART 回归树，这样得到的模型就称为梯度提升树（Gradient Boosting Decision Tree，GBDT）。GBDT 的基决策树是弱学习器，通常深度较小，一般不会超过 5，叶子节点的数量不会超过 10。GBDT 的关键是如何基于 $\{(x_i, r_{ti})\}_{i=1}^{m}$ 生成 CART 回归树 $h_t(x)$。

第 t 轮的第 i 个样本的损失函数的负梯度为

$$r_{ti} = -\left[\frac{\partial L(y, f(x_i))}{\partial f(x_i)} \right]_{f(x)=f_{t-1}(x)}$$

利用 (x_i, r_{ti}) $(i = 1, 2, ..m)$，我们可以拟合一棵 CART 回归树，得到了第 t 颗回归树，其对应的叶节点区域 R_{tj}，$j = 1, 2, ..., J$，其中 J 为叶子节点的个数。

针对每一个叶子节点里的样本，我们求出使损失函数最小，也就是拟合叶子节点最好的的输出值 c_{tj} 如下：

$$c_{tj} = \underset{c}{argmin} \sum_{x_i \in R_{tj}} L(y_i, f_{t-1}(x_i) + c)$$

这样我们就得到了本轮的决策树拟合函数如下：

$$h_t(x)=\sum_{j=1}^{J}c_{tj}I(x\in R_{tj})$$

从而本轮最终得到的强学习器的表达式如下：

$$f_t(x)=f_{t-1}(x)+\sum_{j=1}^{J}c_{tj}I(x\in R_{tj})$$

通过损失函数的负梯度来拟合，我们找到了一种通用的拟合损失误差的办法，这样无轮是分类问题还是回归问题，我们通过其损失函数的负梯度的拟合，就可以用 GBDT 来解决我们的分类回归问题。区别仅仅在于损失函数不同导致的负梯度不同而已。

对于分类问题，常用损失函数有

- 二分类问题，对数似然损失函数

$$L\left(y,\ f\left(x\right)\right)\ =\ log\left(1+\ exp\left(-yf\left(x\right)\right)\right)\ ,\ y\in\{-1,+1\}$$

则此时的负梯度误差为

$$r_{ti}=-\left[\frac{\partial L(y,f(x_i)))}{\partial f(x_i)}\right]_{f(x)=f_{t-1}(x)}=y_i/(1+exp(y_i f(x_i)))$$

对于生成的决策树，我们各个叶子节点的最佳残差拟合值为

$$c_{tj}=\underset{c}{argmin}\sum_{x_i\in R_{tj}}log(1+exp(y_i(f_{t-1}(x_i)+c)))$$

由于上式比较难求出最优值，一般使用近似值代替

$$c_{tj}=\sum_{x_i\in R_{tj}}r_{ti}\Big/\sum_{x_i\in R_{tj}}|r_{ti}|(2-|r_{ti}|)$$

- K 类分类（ $K>2$ ）问题的对数似然损失函数为：

$$L(y,f(x))=-\sum_{k=1}^{K}y_k log\ p_k(x)$$

其中如果样本输出类别为 k ，则 $y_k=1$。第 k 类的概率 $p_k(x)$ 的表达式为：

$$p_k(x)=exp(f_k(x))\Big/\sum_{l=1}^{K}exp(f_l(x))$$

结合上面两式，我们可以计算出第 t 轮的第 i 个样本对应类别 l 的负梯度误差为

$$r_{til}=-\left[\frac{\partial L(y,f(x_i)))}{\partial f(x_i)}\right]_{f_k(x)=f_{l,t-1}(x)}=y_{il}-p_{l,t-1}(x_i)$$

观察上式可以看出，其实这里的误差就是样本 i 对应类别 l 的真实概率和 $t-1$ 轮预测概率的差值。

对于生成的决策树，我们各个叶子节点的最佳残差拟合值为

$$c_{tjl}=\underset{c_{jl}}{argmin}\sum_{i=0}^{m}\sum_{k=1}^{K}L(y_k,f_{t-1,l}(x)+\sum_{j=0}^{J}c_{jl}I(x_i\in R_{tj})$$

由于上式比较难求出最优值，我们一般使用近似值代替

$$c_{tjl}=\frac{K-1}{K}\frac{\sum_{x_i\in R_{tjl}}r_{til}}{\sum_{x_i\in R_{til}}|r_{til}|(1-|r_{til}|)}$$

对于回归算法，常用损失函数有：

- 误差平方：

$$L(y, f(x)) = \frac{1}{2}(y - f(x))^2$$

对应负梯度误差为：

$$y_i - f(x_i)$$

即负梯度误差为目标值减去预测值，通常称为残差。因此，下一棵回归树就是在拟合这个残差。

- 绝对误差：

$$L\left(y, f\left(x\right)\right) = \left|y - f\left(x\right)\right|$$

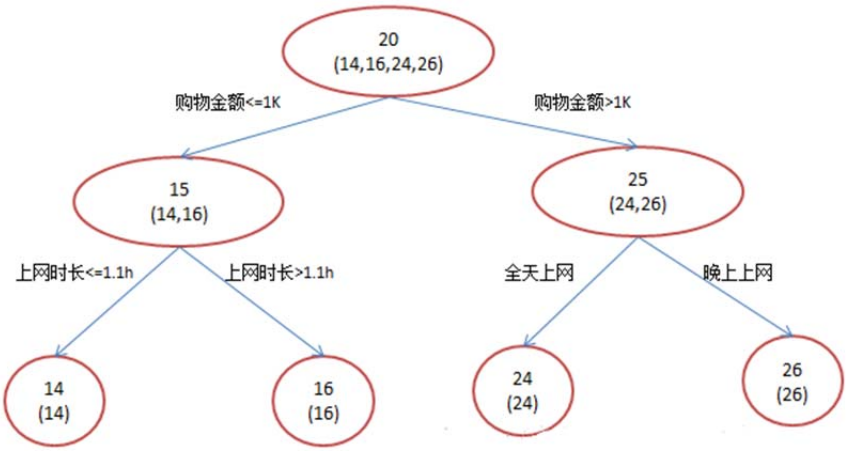
对应负梯度误差为：

$$sign\left(y_i - f\left(x_i\right)\right)$$

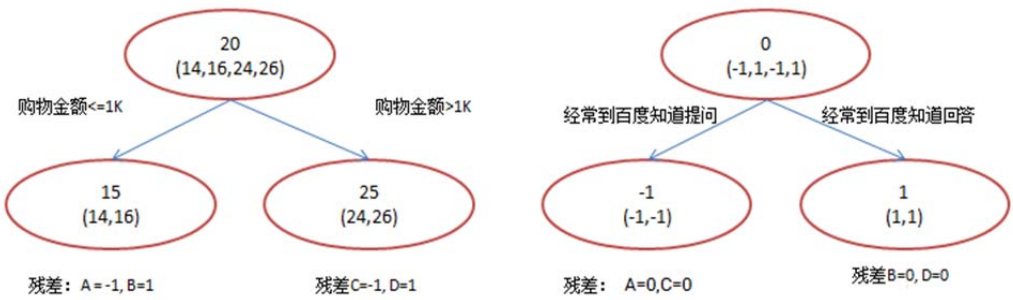
另外还有 Huber 损失函数和分位数损失函数等，主要用于健壮回归，能够就是减少异常点对损失函数的影响。

下面利用一个实例说明 GBDT 的基于残差拟合的回归算法。

假设我们现在有一个训练集，训练集只有 4 个人，A, B, C, D，他们的年龄分别是 14, 16, 24, 26。其中 A、B 分别是高一和高三学生；C, D 分别是应届毕业生和工作两年的员工。样本中有购物金额、上网时长、经常到百度知道提问等特征。如果是用一棵传统的 CART 回归树来训练，会得到如下图结果：



现在我们使用 GBDT 来做这件事，由于数据太少，我们限定叶子结点最多有两个，即每棵树都只有一个分枝，并且限定只学两棵树。我们会得到如下图所示结果：



在第一棵树分支，由于 A,B 年龄较为相近，C,D 年龄较为相近，他们被分为两拨，每拨用平均年龄作为预测值。此时计算残差，所以 A 的残差就是$16-15=1$（注意，A 的预测值是指前面所有树累加的和，这里前面只有一棵树所以直接是 15，如果还有树则需要都累加起来作为 A 的预测值）。进而得到 A,B,C,D 的残差分别为$-1,1,-1,1$。然后我们拿残差替代 A,B,C,D 的原值，即（A，-1岁）、（B，1 岁）、（C，-1岁）、（D, 1 岁），到第二棵树去学习，如果我们的预测值和它们的残差相等，则只需把第二棵树的结论累加到第一棵树上就能得到真实年龄了。第二棵树只有两个值 1 和-1，直接分成两个结点，此时所有人的残差都是 0，停止学习。

CART 回归树为了达到 100%精度使用了 3 个特征（上网时长、上网时段、网购金额），GBDT 虽然用了两棵树 ，但其实只用了 2 个特征就搞定了，泛化能力更强。

实际应用中 GBDT 还要在每棵树的输出值乘以一个很小的系数（通常小于 0.1）：

$$f_t(x)=f_{t-1}(x)+\nu\cdot\gamma_t h_t(x),\quad 0<\nu\le 1$$

其中ν称为收缩因子(shrinkage)，作用相当于步长或学习率。经验证明，适当的收缩因子参数能减少过拟合。

4.3 Xgboost

xgboost，全称为 eXtreme Gradient Boosting（极端梯度提升），是基于 GBDT 进行改进的算法。GBDT 在优化时只用到一阶导数信息，xgboost 则对损失函数进行了二阶泰勒展开，同时用到了损失函数的一阶和二阶导数信息，并增加了正则项，正则项里包含了树的叶子节点个数、每个叶子节点上输出值的 L2 模的平方和，控制了模型的复杂程度，相比 GBDT，xgboost 在训练的过程中不容易陷入过拟合。xgboost 能自动利用单机 CPU 的多核进行并行计算，很好地支持稀疏数据。xgboost 因为出众的效率与较高的预测准确度在数据分析比赛中得到大量的运用。工业界规模方面，xgboost 的分布式版本有广泛的可移植性，支持在 YARN, MPI, Sungrid Engine 等各个平台上面运行，并且保留了单机并行版本的各种优化，使得它可以很好地解决于工业界规模的问题。