# 1. AdaBoost算法

## 1.1. AdaBoost算法

- 输入：训练集$T=\{(x_1,y_1),(x_2,y_2),\cdots,(x_N,y_N)\},x_i\in\mathcal{X}\subseteq\mathbf{R^n},\mathbf{y_i}\in\{-\mathbf{1},+\mathbf{1}\}$
- 输出：最终分类器$G(x)$
- 初始化训练数据的分布$D_1=(w_{11},\cdots,w_{1i},\cdots,w_{1N}),w_{1i}=\frac{1}{N}$
- 对于$m=1,2,\cdots,M$
  - 基本分类器 $G_m(x):\mathcal{X}\rightarrow\{-1,+1\}$

- 计算$G_m(x)$在训练数据集上的分类误差率：$e_m = \sum_{i=1}^{N} P\{G_m(x_i) \neq y_i\} = \sum_{i=1}^{N} w_{mi} I\{G_m(x_i) \neq y_i\}$
- 计算$G_m(x)$的系数：$\alpha_m = \frac{1}{2} \log \frac{1-e_m}{e_m}$
- 更新训练数据集的权值分布：

$$
\begin{aligned}
D_{m+1} &= (w_{m+1,1}, \cdots, w_{m+1,i}, \cdots, w_{m+1,N}) \\
w_{m+1,i} &= \frac{w_{mi}}{Z_m} \exp\{-\alpha_m y_i G_m(x_i)\} = \begin{cases} \frac{w_{mi}}{Z_m} e^{-\alpha_m} & \text{if } G_m(x_i) = y_i \\ \frac{w_{mi}}{Z_m} e^{\alpha_m} & \text{if } G_m(x_i) \neq y_i \end{cases} \\
Z_m &= \sum_{i=1}^{N} w_{mi} \exp\{-\alpha_m y_i G_m(x_i)\}
\end{aligned}
$$

- 构建基本分类器的线性组合：$f(x) = \sum_{m=1}^{M} \alpha_m G_m(x)$
- 得到最终分类器：$G(x) = \text{sign}(f(x)) = \text{sign}\{\sum_{m=1}^{M} \alpha_m G_m(x)\}$

当$e_m \leq \frac{1}{2}$时，$\alpha_m \geq 0$，且$\alpha_m$随着$e_m$的减小而增大，所以分类误差率越小的基本分类器在最终分类器中的作用越大。

# 1.2. 前向分布算法

**AdaBoost算法另一个解释，可认为AdaBoost算法是模型为加法模型、损失函数为指数函数、学习算法为前向分布算法时的二分类学习方法。**

- 加法模型：$f(x) = \sum_{m=1}^{M} \beta_m b(x; \gamma_m)$
- $b(x; \gamma_m)$为基函数
- $\gamma_m$为基函数的参数
- $\beta_m$为基函数的系数
- 在给定训练数据及损失函数$L(y, f(x))$的条件下，学习加法模型$f(x)$成为**经验风险极小化**即**损失函数极小化**问题：$\min_{\beta_m, \gamma_m} \sum_{i=1}^{N} L\big(y_i, \sum_{m=1}^{M} \beta_m b(x_i; \gamma_m)\big)$

前向分布算法求解这一损失函数优化问题的想法是：从前向后，每一步只学习一个基函数及其系数，逐步逼近优化目标函数。具体地，每步只需优化损失函数$\min_{\beta,\gamma} \sum_{i=1}^{N} L\big(y_i, \beta b(x_i; \gamma)\big)$

**前向分布算法：**

- 输入：训练集$T = \{(x_1, y_1), (x_2, y_2), \cdots, (x_N, y_N)\}, x_i \in \mathcal{X} \subseteq \mathbf{R^n}, \mathbf{y_i} \in \{-\mathbf{1}, +\mathbf{1}\}$；损失函数$L(y, f(x))$；基函数集$\{b(x; \gamma)\}$
- 输出：加法模型$f(x)$
- 初始化$f_0(x) = 0$
- 对于$m = 1, 2, \cdots, M$
  - 极小化损失函数$(\beta_m, \gamma_m) = \arg\min_{\beta,\gamma} \sum_{i=1}^{N} L\big(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma)\big)$得到参数$\beta_m, \gamma_m$
  - 更新$f_m(x) = f_{m-1}(x) + \beta_m b(x; \gamma_m)$

前向分布算法将同时求解从$m = 1$到$M$所有参数$\beta_m, \gamma_m$的优化问题简化为逐次求解各个$\beta_m, \gamma_m$的优化问题。

# 1.3. 前向分布算法与AdaBoost

定理：AdaBoost算法是前向分布加法算法的特例。这时，模型是由基本分类器组成的加法模型，损失函数是指数函数。

**证明前向分布算法的损失函数是指数损失函数$L\big(y, f(x)\big) = \exp\big(-yf(x)\big)$时，其学习的具体操作等价于AdaBoost算法学习的具体操作。**

- 假设经过$m-1$轮迭代前向分布算法，已经得到$f_{m-1}(x)=f_{m-2}(x)+\alpha_{m-1}G_{m-1}(x)=\alpha_1G_1(x)+\cdots+\alpha_{m-1}G_{m-1}(x)$.
- 在第$m$轮迭代得到$\alpha_m,G_m(x),f_m(x)$
- 目标是使前向分布算法得到的$\alpha_m,G_m(x)$使$f_m(x)$在训练数据集$T$上的指数损失最小，即

$$\begin{aligned}\big(\alpha_m,G_m(x)\big)&=\arg\min_{\alpha,G}\sum_{i=1}^{N}\exp\big[-y_i\big(f_{m-1}(x_i)+\alpha G(x_i)\big)\big]\\&=\arg\min_{\alpha,G}\sum_{i=1}^{N}\bar{w}_{mi}\exp[-y_i\alpha G(x_i)]\\\bar{w}_{mi}&=\exp[-y_if_{m-1}(x_i)]\end{aligned}\tag{1}$$

  $\bar{w}_{mi}$既不依赖$\alpha$也不依赖于$G$，所以与最小化无关。
  $\bar{w}_{mi}$依赖于$f_{m-1}(x)$，随着每一轮迭代而发生改变。

**证明使式(1)达到最小的$\alpha_m^*,G_m^*(x)$就是AdaBoost算法所得到的$\alpha_m,G_m(x)$**

- step1 求解$G_m^*(x)$
  对任意$\alpha>0$，使式(1)最小的$G(x)$由下式得到：

$$\begin{aligned}G_m^*(x)&=\arg\min_{G}\sum_{i=1}^{N}\bar{w}_{mi}I\big(y_i\neq G(x_i)\big)\\\bar{w}_{mi}&=\exp[-y_if_{m-1}(x_i)]\end{aligned}$$

- step2 求解$\alpha_m^*$

$$\sum_{i=1}^{N}\bar{w}_{mi}\exp[-y_i\alpha G(x_i)]=\sum_{y_i=G_m(x_i)}\bar{w}_{mi}e^{-\alpha}+\sum_{y_i\notin G_m(x_i)}\bar{w}_{mi}e^{\alpha}$$

# 2. 决策树

- 设训练数据集为$D$，$|D|$表示其样本容量，即样本个数
- 设有$K$个类$C_k,k=1,2,\cdots,K$，$|C_k|$为属于类$C_k$的样本个数，$\sum_{k=1}^{K}|C_k|=|D|$
- 设特征$A$由$n$个不同的取值$\{a_1,a_2,\cdots,a_n\}$，根据特征$A$的取值将$D$划分为$n$个子集$D_1,D_2,\cdots,D_n$，$|D_i|$为$D_i$的样本个数，$\sum_{i=1}^{n}|D_i|=|D|$
- 记子集$D_i$中属于类$C_k$的样本集合为$D_{ik}$，即$D_{ik}=D_i\cap C_k$，$|D_{ik}|$为$D_{ik}$的样本个数
- 数据集$D$的经验熵$H(D)=-\sum_{k=1}^{K}\frac{|C_k|}{|D|}\log_2\frac{|C_k|}{|D|}$
- 特征$A$对数据集$D$的经验条件熵$H(D|A)=\sum_{i=1}^{n}\frac{|D_i|}{|D|}H(D_i)=-\sum_{i=1}^{n}\frac{|D_i|}{|D|}\sum_{k=1}^{K}\frac{|D_{ik}|}{|D_i|}\log_2\frac{|D_{ik}|}{|D_i|}$
- 信息增益$g(D,A)=H(D)-H(D|A)$
- 特征$A$对数据集$D$的信息增益比$g_R(D,A)=\frac{g(D,A)}{H_A(D)},H_A(D)=-\sum_{i=1}^{n}\frac{|D_i|}{|D|}\log_2\frac{|D_i|}{|D|}$，$n$是特征$A$取值的个数

| $A_1$年龄 | 类别 | 数量 |
|---|---|---|
| 青年1 | 是 | 2 |
| 青年1 | 否 | 3 |

| $A_1$年龄 | 类别 | 数量 |
| --- | --- | --- |
| 中年2 | 是 | 3 |
| 中年2 | 否 | 2 |
| 老年3 | 是 | 4 |
| 老年3 | 否 | 1 |

$$
\begin{aligned}
H(D) &= -\frac{9}{15}\log_2\frac{9}{15} - \frac{6}{15}\log_2\frac{6}{15} = 0.971 \\
g(D, A_1) &= H(D) - [\frac{5}{15}H(D_1) + \frac{5}{15}H(D_2) + \frac{5}{15}H(D_3)] \\
&= 0.971 - [\frac{5}{15}(-\frac{2}{5}\log_2\frac{2}{5} - \frac{3}{5}\log_2\frac{3}{5}) + \frac{5}{15}(-\frac{3}{5}\log_2\frac{3}{5} - \frac{2}{5}\log_2\frac{2}{5}) + \frac{5}{15}(-\frac{4}{5}\log_2\frac{4}{5} - \frac{1}{5}\log_2\frac{1}{5})] \\
&= 0.971 - 0.888 = 0.083
\end{aligned}
$$

$$
\begin{aligned}
Gini(D, A_1 = 1) &= \frac{5}{15}(2 \times \frac{2}{5} \times (1 - \frac{2}{5})) + \frac{10}{15}(2 \times \frac{7}{10} \times (1 - \frac{7}{10})) = 0.44 \\
Gini(D, A_1 = 2) &= \frac{5}{15}(2 \times \frac{3}{5} \times (1 - \frac{3}{5})) + \frac{10}{15}(2 \times \frac{6}{10} \times (1 - \frac{6}{10})) = 0.48 \\
Gini(D, A_1 = 3) &= \frac{5}{15}(2 \times \frac{4}{5} \times (1 - \frac{4}{5})) + \frac{10}{15}(2 \times \frac{5}{10} \times (1 - \frac{5}{10})) = 0.44
\end{aligned}
$$

| $A_2$有工作 | 类别 | 数量 |
| --- | --- | --- |
| 是1 | 是 | 5 |
| 是1 | 否 | 0 |
| 否2 | 是 | 4 |
| 否2 | 否 | 6 |

$$
\begin{aligned}
g(D, A_2) &= H(D) - [\frac{5}{15}H(D_1) + \frac{10}{15}H(D_2)] \\
&= 0.971 - [\frac{5}{15}(-\frac{5}{5}\log_2\frac{5}{5} - \frac{0}{5}\log_2\frac{0}{5}) + \frac{10}{15}(-\frac{4}{10}\log_2\frac{4}{10} - \frac{6}{10}\log_2\frac{6}{10})] \\
&= 0.324 \\
Gini(D, A_2 = 1) &= \frac{5}{15}(2 \times \frac{5}{5} \times (1 - \frac{5}{5})) + \frac{10}{15}(2 \times \frac{4}{10} \times (1 - \frac{4}{10})) = 0.32
\end{aligned}
$$

| $A_3$有自己的房子 | 类别 | 数量 |
| --- | --- | --- |
| 是1 | 是 | 6 |
| 是1 | 否 | 0 |
| 否2 | 是 | 3 |
| 否2 | 否 | 6 |

$$
\begin{aligned}
g(D, A_3) &= H(D) - [\frac{6}{15}H(D_1) + \frac{9}{15}H(D_2)] \\
&= 0.971 - [\frac{6}{15}(-\frac{6}{6}\log_2\frac{6}{6} - \frac{0}{6}\log_2\frac{0}{6}) + \frac{9}{15}(-\frac{3}{9}\log_2\frac{3}{9} - \frac{6}{9}\log_2\frac{6}{9})] \\
&= 0.420
\end{aligned}
$$

$$
Gini(D, A_3 = 1) = \frac{6}{15}(2 \times \frac{6}{6} \times (1 - \frac{6}{6})) + \frac{9}{15}(2 \times \frac{3}{9} \times (1 - \frac{3}{9})) = 0.27
$$

| $A_4$信贷情况 | 类别 | 数量 |
|---|---|---|
| 一般3 | 是 | 1 |
| 一般3 | 否 | 4 |
| 好2 | 是 | 4 |
| 好2 | 否 | 2 |
| 非常好1 | 是 | 4 |
| 非常好1 | 否 | 0 |

$$
\begin{aligned}
g(D, A_4) &= H(D) - [\frac{5}{15}H(D_1) + \frac{6}{15}H(D_2) + \frac{4}{15}H(D_3)] \\
&= 0.971 - [\frac{5}{15}(-\frac{1}{5}\log_2\frac{1}{5} - \frac{4}{5}\log_2\frac{4}{5}) + \frac{6}{15}(-\frac{4}{6}\log_2\frac{4}{6} - \frac{2}{6}\log_2\frac{2}{6}) + \frac{4}{15}(-\frac{4}{4}\log_2\frac{4}{4} - \frac{0}{4}\log_2\frac{0}{4})] \\
&= 0.971 - 0.608 = 0.363
\end{aligned}
$$

$$
Gini(D, A_4 = 1) = \frac{4}{15}(2 \times \frac{4}{4} \times (1 - \frac{4}{4})) + \frac{11}{15}(2 \times \frac{5}{11} \times (1 - \frac{5}{11})) = 0.36
$$

$$
Gini(D, A_4 = 2) = \frac{6}{15}(2 \times \frac{4}{6} \times (1 - \frac{4}{6})) + \frac{9}{15}(2 \times \frac{5}{9} \times (1 - \frac{5}{9})) = 0.47
$$

$$
Gini(D, A_4 = 3) = \frac{5}{15}(2 \times \frac{1}{5} \times (1 - \frac{1}{5})) + \frac{10}{15}(2 \times \frac{8}{10} \times (1 - \frac{8}{10})) = 0.32
$$

由于特征$A_3$（有自己的房子）的信息增益值最大，所以选择特征$A_3$作为最优特征。
$Gini(D, A_1 = 1) = 0.44, Gini(D, A_1 = 2) = 0.48, Gini(D, A_1 = 3) = 0.44, Gini(D, A_2 = 1) = 0.32, Gini(D, A_3 = 1) = 0.27, Gini(D, A_4 = 1) = 0.36, Gini(D, A_4 = 2) = 0.47, Gini(D, A_4 = 3) = 0.32$
选择特征$A_3$为最优特征，$A_3 = 1$为其最优切分点。

# 2.1. ID3与C4.5

## 2.1.1. 算法

**ID3算法:**

- 输入：训练数据集$D$，特征集$A$，阈值$\epsilon$
- 输出：决策树$T$
- （1）若$D$中所有实例属于同一类$C_k$，则$T$为单结点树，并将类$C_k$作为该结点的类标记，返回$T$；
- （2）若$A = \varnothing$，则$T$为单结点树，并将$D$中实例数最大的类$C_k$作为该结点的类标记，返回$T$
- （3）否则，计算$A$中各特征对$D$的信息增益，选择信息增益最大的特征$A_g$
- （4）若$A_g$的信息增益小于阈值$\epsilon$，则置$T$为单结点树，并将$D$中实例数最大的类$C_k$作为该结点的类标记，返回$T$
- （5）否则，对$A_g$的每一可能值$a_i$，依$A_g = a_i$将$D$分割为若干非空子集$D_i$，将$D_i$中实例数最大的类作为标记，构建子结点，由结点及其子结点构成树$T$，返回$T$
- （6）对于第$i$个子结点，以$D_i$为训练集，以$A - \{A_g\}$为特征集，递归地调用步（1）～（5），得到子树$T_i$，返回$T_i$

**C4.5算法：** 将ID3中的「信息增益」改为「信息增益比」来选择特征。

## 2.1.2. 决策树剪枝

- 「决策树生成」只考虑了通过提高信息增益（或信息增益比）对训练数据进行更好的拟合
- 「决策树剪枝」通过优化损失函数，还考虑了减小模型复杂度
- 「决策树生成」学习局部的模型
- 「决策树剪枝」学习整体的模型

设树$T$的叶结点个数为$|T|$，$t$是树$T$的叶结点，该叶结点有$N_t$个样本点，其中$k$类的样本点有$N_{tk}, k=1,2,\cdots,K$个，$H_t(T)$为叶结点$t$上经验熵，$\alpha \geq 0$为参数，则决策树学习的损失函数可以定义为$C_\alpha = \sum_{t=1}^{|T|} N_t H_t(T) + \alpha |T|$，其中经验熵为$H_t(T) = -\sum_k \frac{N_{tk}}{N_t} \log \frac{N_{tk}}{N_t}$

- $C(T) = \sum_{t=1}^{|T|} N_t H_t(T) = -\sum_{t=1}^{|T|} \sum_{k=1}^{K} N_{tk} \log \frac{N_{tk}}{N_t}$
- $C_\alpha(T) = C(T) + \alpha |T|$
- $C(T)$表示模型对训练数据的预测误差，即模型与训练数据的拟合程度
- $|T|$表示模型复杂度，参数$\alpha \geq 0$控制两者之间的影响
- 较大的$\alpha$促使选择较简单的模型（树），较小的$\alpha$促使选择较复杂的模型（树）
- 剪枝，就是当$\alpha$确定时，选择损失函数最小的模型，即损失函数最小的子树
- 当$\alpha$确定时，子树越大，往往与训练数据拟合的越好，但是模型的复杂度就越高；相反，子树越小，模型的复杂度就越低，但是往往与训练数据的拟合不好

树的剪枝算法：

- 输入：生成算法产生的整个树$T$，参数$\alpha$
- 输出：修剪后的子树$T_\alpha$
- （1）计算每个结点的经验熵
- （2）递归地从树的叶节点向上回缩
- （3）设一组叶节点回缩到其父结点之前与之后的整体树分别为$T_B$与$T_A$，其对应的损失函数值分别是$C_\alpha(T_B)$与$C_\alpha(T_A)$，若$C_\alpha(T_A) \leq C_\alpha(T_B)$，则进行剪枝，即将父结点变为新的叶结点
- 返回（2），直至不能继续为止，得到损失函数最小的子树$T_\alpha$

# 2.2. CART

classification and regression tree, CART

- 回归树，平方误差最小化标准
- 分类树，基尼指数最小化标准

## 2.2.1. 最小二乘回归树生成算法(回归)

- 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$\min_{j,s} = [\min_{c_1} \sum_{x_i \in R_1(j,s)} (y_i - c_1)^2 + \min_{c_2} \sum_{x_i \in R_2} (y_i - c_2)^2]$，遍历变量，对固定的切分变量$j$扫描切分点$s$，选择使式$\min_{j,s}$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：$R_1(j,s) = \{x|x^{(j)} \leq s\}, R_2(j,s) = \{x|x^{(j)} > s\}, \hat{c}_m = \frac{1}{N_m} \sum_{x_i \in R_m(j,s)} y_i, x \in R_m, m=1,2$
- （3）继续对两个子区域调用步骤(1)(2)，直至满足停止条件。
- 将输入空间划分为$M$个区域$R_1, R_2, \cdots, R_M$，生成决策树$f(x) = \sum_{m=1}^{M} \hat{c}_m I(x \in R_m)$。

## 2.2.2. CART生成算法(分类)

- 分类问题中，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$(p)=\sum_{k=1}^{K}p_k(1-p_k)=1-\sum_{k=1}^{K}p_k^2$
- 二分类，Gini$(p)=2p(1-p)$
- 对于给定的样本集合$D$，Gini$(D)=1-\sum_{k=1}^{K}(\frac{|C_k|}{|D|})^2$，$C_k$是$D$中属于第$k$类的样本子集，$K$是类的个数；基尼指数Gini$(D)$，表示集合$D$的不确定性
- 集合$D$根据特征$A$是否取某一可能值$a$被分割成$D_1$和$D_2$两部分，则在特征$A$的条件下，集合$D$的基尼指数定义为Gini$(D,A)=\frac{|D_1|}{|D|}Gini(D_1)+\frac{|D_2|}{|D|}Gini(D_2)$；基尼指数Gini$(D,A)$，表示经$A=a$分割后集合$D$的不确定性
- 基尼指数值越大，样本集合的不确定性也就越大

CART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
- 根据训练数据集，从根结点开始，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$D$分割成$D_1$和$D_2$两部分，计算Gini$(D,A)$。
- （2）在所有可能的特征$A$及它们所有可能的切分点$a$中，选择基尼指数最小的特征及其对应的切分点作为最优特征与最优切分点。依最优特征与最优切分点，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结点中去。
- 对两个子结点递归地调用(1)(2)，直至满足停止条件。
- 生成CART决策树。
- 算法停止的条件是：结点中的样本个数小于预定阈值，或样本集的基尼指数小于预定阈值（样本基本属于同一类），或者没有更多特征。

## 2.2.3. CART剪枝算法

在剪枝过程中，子树损失函数为$C_\alpha=C(T)+\alpha|T|$

- $T$为任意子树
- $C(T)$为对训练数据的预测误差（如基尼指数）
- $|T|$为子树对叶结点个数
- $C_\alpha(T)$为参数为$\alpha$时的子树$T$的整体损失
- $g(t)=\frac{C(t)-C(T_t)}{|T_t|-1}$表示剪枝后整体损失函数减少的程度

CART剪枝算法：

- 输入：CART算法生成的决策树$T_0$
- 输出：最优决策树$T_\alpha$
- （1）设$k=0,T=T_0$
- （2）设$\alpha=+\infty$
- （3）自下而上地对各内部结点$t$计算$C(T_t),|T_t|$，以及$g(t)=\frac{C(t)-C(T_t)}{|T_t|-1},\alpha=\min(\alpha,g(t))$，$T_t$表示以$t$为根结点的子树，$C(T_t)$是对训练数据的预测误差，$|T_t|$是$T_t$对叶结点个数
- （4）对$g(t)=\alpha$的内部结点$t$进行剪枝，并对叶结点$t$以多数表决法决定其类，得到树$T$
- （5）设$k=k+1,\alpha_k=\alpha,T_k=T$
- （6）如果$T_k$不是由根结点及两个叶结点构成的树，则回到步骤(2)；否则令$T_k=T_n$
- （7）采用交叉验证法在子树序列$T_0,T_1,\cdots,T_n$中选取最优子树$T_\alpha$

# 3. GBDT

## 3.1. Algorithm1 Gradient_Boost

$$
\begin{aligned}
&F_0(x) = \arg\min_{\rho} \sum_{i=1}^{N} L(y_i, \rho) \\
&\text{For } m = 1 \text{ to } M \text{ do:} \\
&\tilde{y}_i = -\Big[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\Big]_{F(x)=F_{m-1}(x)}, i = 1, N \\
&\alpha_m = \arg\min_{\alpha,\beta} \sum_{i=1}^{N} [\tilde{y}_i - \beta h(x_i;\alpha)]^2 \\
&\rho_m = \arg\min_{\rho} \sum_{i=1}^{N} L(y_i, F_{m-1}(x_i) + \rho h(x_i;\alpha_m)) \\
&F_m(x) = F_{m-1}(x) + \rho_m h(x;\alpha_m) \\
&\text{endFor} \\
&\text{end Algorithm}
\end{aligned}
$$

函数$h(x;\alpha)$是weaker learner或base learner，通常是分类树。

## 3.2. Algorithm2 LeastSquares_Boost

$$
\begin{aligned}
&L(y, F) = \frac{1}{2}(y - F)^2 \\
&F_0(x) = \bar{y} \\
&\text{For } m = 1 \text{ to } M \text{ do:} \\
&\tilde{y}_i = y_i - F_{m-1}(x_i) \\
&(\rho_m, \alpha_m) = \arg\min_{\alpha,\rho} \sum_{i=1}^{N} [\tilde{y}_i - \rho h(x_i;\alpha)]^2 \\
&F_m(x) = F_{m-1}(x) + \rho_m h(x;\alpha_m) \\
&\text{endFor} \\
&\text{end Algorithm}
\end{aligned}
$$

## 3.3. Algorithm3 LeastAbsoluteDeviation_TreeBoost

$L(y,F)=|y-F|$

$\widetilde{y}_i=\text{sign}\{y_i-F_{m-1}(x_i)\}=-[\frac{\partial L(y_i,F(x_i))}{\partial F(x_i)}]_{F(x)=F_{m-1}(x)}$

$F_0(x)=\text{median}\{y_i\}_1^N$

For $m=1$ to $M$ do:

$\widetilde{y}_i=\text{sign}\{y_i-F_{m-1}(x_i)\},i=1,N$

$\{R_{jm}\}_1^J=\text{J-terminal node tree}(\{\widetilde{y}_i,x_i\}_1^N)$

$\gamma_{jm}=\text{median}_{x_i\in R_{jm}}\{y_i-F_{m-1}(x_i)\},j=1,J$

$F_m(x)=F_{m-1}(x)+\sum_{j=1}^J\gamma_{jm}1(x\in R_{jm})$

endFor

end Algorithm

## 3.4. Algorithm4 M_TreeBoost

$L(y,F)=\begin{cases}\frac{1}{2}(y-F)^2,|y-F|\le\delta\\ \delta(|y-F|-\delta/2),|y-F|>\delta\end{cases}$

$F_0(x)=\text{median}\{y_i\}_1^N$

For $m=1$ to $M$ do:

$r_{m-1}(x_i)=y_i-F_{m-1}(x_i),i=1,N$

$\delta_m=\text{quantile}_\alpha\{|r_{m-1}(x_i)|\}_1^N$

$\widetilde{y}_i=\begin{cases}r_{m-1}(x_i),|r_{m-1}(x_i)|\le\delta_m\\ \delta_m\cdot\text{sign}(r_{m-1}(x_i)),|r_{m-1}(x_i)|>\delta_m\end{cases}$

$\{R_{jm}\}_1^J=\text{J-terminal node tree}(\{\widetilde{y}_i,x_i\}_1^N)$

$\tilde{r}_{jm}=\text{median}_{x_i\in R_{jm}}\{r_{m-1}(x_i)\},j=1,J$

$\gamma_{jm}=\tilde{r}_{jm}+\frac{1}{N_{jm}}\sum_{x_i\in R_{jm}}\text{sign}(r_{m-1}(x_i)-\tilde{r}_{jm})\cdot\min(\delta_m,\text{abs}(r_{m-1}(x_i)-\tilde{r}_{jm})),j=1,J$

$F_m(x)=F_{m-1}(x)+\sum_{j=1}^J\gamma_{jm}1(x\in R_{jm})$

endFor

end Algorithm

## 3.5. Algorithm5 L_K_TreeBoost(two-class logistic)

$$L(y,F)=\log(1+\exp(-2yF)),y\in\{-1,1\}$$
$$F(x)=\frac{1}{2}\log[\frac{\Pr(y=1|x)}{\Pr(y=-1|x)}]$$
$$F_0(x)=\frac{1}{2}\log\frac{1+\widetilde{y}}{1-y}$$
For $m=1$ to $M$ do:
$$\widetilde{y}_i=\frac{2y_i}{1+\exp(2y_iF_{m-1}(x_i))},i=1,N$$
$$\{R_{jm}\}_1^J=\text{J-terminal node tree}(\{\widetilde{y}_i,x_i\}_1^N)$$
$$\gamma_{jm}=\sum_{x_i\in R_{jm}}\widetilde{y}_i/\sum_{x_i\in R_{jm}}|\widetilde{y}_i|(2-|\widetilde{y}_i|),j=1,J$$
$$F_m(x)=F_{m-1}(x)+\sum_{j=1}^J\gamma_{jm}1(x\in R_{jm})$$
$$p_+(x)=\hat{\Pr}(y=1|x)=\frac{1}{1+e^{-2F_M(x)}}$$
$$p_-(x)=\hat{\Pr}(y=-1|x)=\frac{1}{1+e^{2F_M(x)}}$$
$$\hat{y}(x)=2\cdot1[c(-1,1)p_+(x)>c(1,-1)p_-(x)]-1$$
$c(\hat{y},y)$ is the cost associated with predicting $\hat{y}$ when the truth is $y$

$$-2F_M(x)=\log[\frac{\Pr(y=-1|x)}{\Pr(y=1|x)}]$$
$$e^{-2F_M(x)}=\frac{\Pr(y=-1|x)}{\Pr(y=1|x)}=\frac{p}{1-p}$$
$$1+e^{-2F_M(x)}=\frac{1-p+p}{1-p}=\frac{1}{1-p}$$

对于二分类的logistic回归问题，在第$m$次迭代的经验损失函数为$\phi(\rho,\alpha)=\sum_{i=1}^N\log[1+\exp(-2y_iF_{m-1}(x_i))\cdot\exp(-2y_i\rho h(x_i;\alpha))]$。
如果$y_iF_{m-1}(x_i)$非常大，则经验损失函数值接近于0，即$(\rho_m,\alpha_m)=\arg\min_{\rho,\alpha}\phi_m(\rho,\alpha)$，这表明对于计算$y_iF_{m-1}(x_i)$非常大的观察值可以从第$m$次迭代中删除。因此，$w_i=\exp(-2y_iF_{m-1}(x_i))$可被视为衡量基于估计$\rho_mh(x;\alpha_m)$的第$i$次观测的影响或权重的测量方法。
另一种衡量在第$m$次迭代中基于估计$\rho_mh(x;\alpha_m)$的第$i$次观测的影响或权重：$w_i=|\widetilde{y}_i|(1-|\widetilde{y}_i|)$，influence trimming删除$w_i<w_{l(\alpha)}$的所有观测值，其中$\sum_{i=1}^{l(\alpha)}w_{(i)}=\alpha\sum_{i=1}^Nw_i$，$\{w_{(i)}\}_1^N$是以升序排序的权重，$\alpha\in[0.05,0.2]$。

## 3.6. Algorithm6 L_K_TreeBoost(multi-class logistic)

$$L(\{y_k,F_k(x)\}_1^K)=-\sum_{k=1}^K y_k\log p_k(x),y_k=1(class=k)\in(0,1),p_k=Pr(y_k=1|x)$$

$$F_k(x)=\log p_k(x)-\frac{1}{K}\sum_{l=1}^K\log p_l(x)$$

$F_{k0}=0,k=1,K$

For $m=1$ to $M$ do:

$$p_k(x)=\exp(F_k(x))/\sum_{l=1}^K\exp(F_l(x)),k=1,K$$

For $k=1$ to $K$ do:

$\widetilde{y}_{ik}=y_{ik}-p_k(x_i),i=1,N$

$\{R_{jkm}\}_{j=1}^J=\text{J-terminal node tree}(\{\widetilde{y}_{ik},x_i\}_1^N)$

$$\gamma_{jkm}=\frac{K-1}{K}\frac{\sum_{x_i\in R_{jkm}}\widetilde{y}_{ik}}{\sum_{x_i\in R_{jkm}}|\widetilde{y}_{ik}|(1-|\widetilde{y}_{ik}|)},j=1,J$$

$$F_{km}(x)=F_{k,m-1}(x)+\sum_{j=1}^J\gamma_{jkm}1(x\in R_{jkm})$$

endFor

endFor

end Algorithm

$$\hat{k}(x)=\arg\min_{1\le k\le K}\sum_{k'=1}^K c(k,k')p_{k'M}(x)$$

## 3.7. 回归树

考虑每个基学习器都是一个有$J$个叶节点的回归树，每个回归树有自己additive形式$h(x;\{b_j,R_j\}_1^J)=\sum_{j=1}^J b_j1(x\in R_j)$，此处$\{R_j\}_1^J$是预测变量$x$的所有可能取值的不相连的区域。这些区域由相关树的叶节点来表示。当条件正确时，指示函数$1(\cdot)$为1，否则为0。

由于区域是不相连的，$h(x;\{b_j,R_j\}_1^J)=\sum_{j=1}^J b_j1(x\in R_j)$等价于预测规则：如果$x\in R_j$，则$h(x)=b_j$。

$F_m(x)=F_{m-1}(x)+\rho_m h(x;\alpha_m)\Rightarrow F_m(x)=F_{m-1}(x)+\rho_m\sum_{j=1}^J b_{jm}1(x\in R_{jm})$，其中$\{R_{jm}\}_1^J$是第$m$次迭代树的叶节点所定义的区域。$b_{jm}=\text{ave}_{x_i\in R_{jm}}\widetilde{y}_i$

# 4. XGBoost

XGBoost中有3个衡量特征重要性的指标：weight、gain、cover

weight：表示在模型中一个特征被选作分裂特征的次数

gain：表示特征在模型中被使用的平均收益，收益通过损失函数的变化度量

cover：表示特征在模型中被使用的平均覆盖率，通过节点的二阶梯度和来度量

## 4.1. tree boosting概述

### 4.1.1. Regularized Learning Objective

给定$m$个特征、样本量为$n$的数据集$D = \{(x_i, y_i)\}, x_i \in \mathbb{R}^m, y_i \in \mathbb{R}$，则集成树模型的预测为$\hat{y}_i = \phi(x_i) = \sum_{k=1}^{K} f_k(x_i), f_k \in \mathcal{F}$，其中$\mathcal{F} = \{f(x) = w_{q(x)}\}(q : \mathbb{R}^m \to T, w \in \mathbb{R}^T)$是CART的空间。

- $q$：每棵树上将样本映射到相应的叶节点上
- $T$：树上的叶节点个数
- 每个$f_k$对应着一棵独立树结构$q$和叶节点权重$w$
- 与决策树不同，每棵回归树上的每个叶节点上包含一个连续分数值，因此使用$w_i$表示第$i$个叶节点上的分数值

$$\mathcal{L}(\phi) = \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k)$$
$$\Omega(f) = \gamma T + \frac{1}{2}\lambda \parallel w \parallel^2$$

- $l$是测量预测值$\hat{y}_i$与真实值$y_i$之间差异的凸损失函数
- $\Omega$项是用来惩罚模型的复杂度的

## 4.1.2. Gradient Tree Boosting

$$f(x) = f(x_0) + f^{'}(x_0)(x - x_0) + \frac{1}{2}f^{''}(x_0)(x - x_0)^2 + \cdots \tag{1}$$
$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) \tag{2}$$
$$\mathcal{L}^{(t)} \simeq \sum_{i=1}^{n} [l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t) \tag{3}$$
$$g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)}) \tag{4}$$
$$h_i = \partial^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)}) \tag{5}$$
$$l(y_i, \hat{y}_i^{(t-1)}) \text{ is constant terms, can be removed} \tag{6}$$
$$\tilde{\mathcal{L}}^{(t)} = \sum_{i=1}^{n} [g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t) \tag{7}$$
$$\text{Define } I_j = \{i | q(x_i) = j\} \text{ as the instance set of leaf } j \tag{8}$$
$$\tilde{\mathcal{L}}^{(t)} = \sum_{i=1}^{n} [g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2 \tag{9}$$
$$= \sum_{j=1}^{T} [(\sum_{i \in I_j} g_i) w_j + \frac{1}{2} (\sum_{i \in I_j} h_i + \lambda) w_j^2] + \gamma T \tag{10}$$
$$\text{For a fixed structure } q(x), \text{ we can compute the optimal weight } w_j^* \text{ of leaf } j \tag{11}$$
$$w_j^* = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda} \tag{12}$$
$$\tilde{\mathcal{L}}^{(t)}(q) = -\frac{1}{2} \sum_{j=1}^{T} \frac{(\sum_{i \in I_j} g_i)^2}{\sum_{i \in I_j} h_i + \lambda} + \gamma T \tag{13}$$

- 上述最后一个公式可作为分数函数，用来衡量树结构$q$的质量
- 该分数类似于评估决策树的impurity score，except that it is derived for a wide range of objective functions

假设$I_L$与$I_R$是节点分裂后的左右节点上的样本集合，$I = I_L \cup I_R$，则分裂后损失函数的减少值为（分裂前损失函数 - 分裂后损失函数，该减少值在实际中被用来评估候选分裂点）：

$$\mathcal{L}_{\text{split}} = \frac{1}{2}[\frac{(\sum_{i\in I_L} g_i)^2}{\sum_{i\in I_L} h_i + \lambda} + \frac{(\sum_{i\in I_R} g_i)^2}{\sum_{i\in I_R} h_i + \lambda} - \frac{(\sum_{i\in I} g_i)^2}{\sum_{i\in I} h_i + \lambda}] - \gamma$$

### 4.1.3. Shrinkage and Column Sub-sampling

除了上述的目标函数中加入正则化，另外两个技巧被用来进一步阻止过拟合问题。

- **shrinkage introduced by Friedman**：shrinkage newly added weights by a factor $\eta$ after each step of tree boosting. Similar to a learning rate in tochastic optimization, shrinkage reduces the influence of each individual tree and leaves space for future trees to improve the model【也即步长$\eta$，也会称为shrinkage】
- **column ( feature ) sub-sampling** (二次抽样)：该技术也被应用于随机森林中；根据反馈，使用column sub-sampling比使用传统的row sub-sampling更会阻止过拟合问题；column sub-sampling同时也提高了在并行计算中的速度

## 4.2. 分裂算法

### 4.2.1. Basic Exact Greedy Algorithm

Algorithm: Exact Greedy Algorithm for Split Finding

- 输入：$I$, instance set of current node
- 输入：$d$, feature dimension
- gain $\leftarrow$ 0
- $G \leftarrow \sum_{i\in I} g_i, H \leftarrow \sum_{i\in I} h_i$
- for $k = 1$ to $m$ do
- $\quad G_L \leftarrow 0, H_L \leftarrow 0$
- $\quad$ for $j$ in sorted $(I, \text{by } x_{jk})$ do
- $\quad\quad G_L \leftarrow G_L + g_j, H_L \leftarrow H_L + h_j$
- $\quad\quad G_R \leftarrow G - G_L, H_R \leftarrow H - H_L$
- $\quad\quad \text{score} \leftarrow \max\{\text{score}, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda}\}$
- $\quad$ end
- end
- 输出：Split with max score

### 4.2.2. Approximate Algorithm

exact greedy algorithm 很强大，因为该算法贪婪地枚举所有可能地分割点。但当数据很大，以至于不能将全部数据读入内存中时，该算法效率低；同样的问题也出现在分布式设置中。
此时可考虑approximate algorithm：

- 算法首先根据特征分布的分位数，提出候选分割点
- 将连续变量根据这些候选分割点映射入不同的分桶内，聚合统计量，基于聚合统计量找到最优的分割点

具体来说，Algorithm: Approximate Algorithm for Split Finding

- for $k$ to $m$ do
- $\quad$ Propose $S_k = \{s_{k1}, s_{k2}, \cdots, s_{kl}\}$ by percentiles on feature $k$
- $\quad$ Propose can be done per tree (global), or per split (local)
- end
- for $k = 1$ to $m$ do
- $\quad G_{kv} \leftarrow= \sum_{j\in\{j|s_{k,v}\geq x_{jk} > s_{k,v-1}\}} g_j$
- $\quad H_{kv} \leftarrow= \sum_{j\in\{j|s_{k,v}\geq x_{jk} > s_{k,v-1}\}} h_j$

- end
- Follow same step as in previous section to find max score only among proposed splits

算法有两个变体，区别在于何时计算变量分割点：

- global方法，构建树结构的初始阶段就给出所有的候选分割点，后面所有节点的分割点的查找都基于该候选分割点
- local方法，每次分割后，都重新给出候选分割点
- 相较于local方法，global方法需要**更少的候选分割点步骤**
- 但，通常global方法需要**更多的候选分割点**，因为在每次分割后，候选分割点会变得不精确；local方法在每次分割后，都对候选分割点进行精确，更适合较深的树

## 4.2.3. 切分点的选取 Weighted Quantile Sketch

let multi-set $D_k = \{(x_{1k}, h_1), (x_{2k}, h_2), \cdots, (x_{nk}, h_n)\}$ represent the k-th feature values and second order gradient statistics of each training instances.
define a rank function $r_k : \mathbb{R} \to [0, +\infty)$ as

$$r_k(z) = \frac{1}{\sum_{(x,h)\in D_k} h} \sum_{(x,h)\in D_k, x<z} h$$

which represents the proportion of instances whose feature value $k$ is smaller than $z$.
The goal is to find candidate split points $\{s_{k1}, s_{k2}, \cdots, s_{kl}\}$, such that

$$|r_k(s_{k,j}) - r_k(s_{k,j+1})| < \epsilon, s_{k1} = \min_i x_{ik}, s_{kl} = \max_i x_{ik}$$

$\epsilon$是近似因子，直观上大体可理解为约有$\frac{1}{\epsilon}$个候选点。
根据下面公式来理解，$h_i$ 为什么是权重?

$$\begin{aligned}
\hat{\mathcal{L}}^{(t)} &= \sum_{i=1}^{n} [g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t) \\
&= \sum_{i=1}^{n} \frac{1}{2} h_i [2\frac{g_i}{h_i} f_t(x_i) + f_t^2(x_i)] + \Omega(f_t) \\
&= \sum_{i=1}^{n} \frac{1}{2} h_i [\frac{g_i^2}{h_i^2} + 2\frac{g_i}{h_i} f_t(x_i) + f_t^2(x_i)] + \Omega(f_t) + \text{constant} \\
&= \sum_{i=1}^{n} \frac{1}{2} h_i (f_t(x_i) - (-\frac{g_i}{h_i}))^2 + \Omega(f_t) + \text{constant}
\end{aligned}$$

which is exactly weighted squared loss with labels $\frac{g_i}{h_i}$ and weights $h_i$

- 对于大的数据集，很有必要去寻找满足条件限制的候选分割点
- 当每个样本有同样的权重时，现有的算法quantile sketch能够解决该问题
- 但没有能够解决加权数据集的quantile sketch
- 现有的近似算法，要么对数据集的随机子集进行排序(有可能失败)，或者启发式地(没有理论保证)
- 为此，创新地提出 分布式加权的quantile sketch，能够在理论证明的保证下来处理加权数据

## 4.2.4. Sparsity-aware Split Finding

现实问题中，输入$X$很可能是稀疏的，导致稀疏的可能原因如下：

- 数据中出现缺失值
- 统计出许多0值
- 使用one-hot做特征工程

很有必要让算法知道数据中的稀疏部分，为此在每棵树节点上加了一个默认分支。
当在稀疏矩阵$X$中出现缺失值，样本就被划分到默认的分支上

Algorithm: Sparsity-aware Split Finding

- 输入：$I,\ \text{instance set of current node}$
- 输入：$I_k = \{i \in I | x_{ik} \neq \text{missing}\}$
- 输入：$d,\ \text{feature dimension}$
- also applies to the approximate setting, only collect statistics of non-missing entries into buckets
- gain $\leftarrow 0$
- $G \leftarrow \sum_{i \in I} g_i, H \leftarrow \sum_{i \in I} h_i$
- for $k = 1$ to $m$ do
- 　　// enumerate missing value goto right
- 　　$G_L \leftarrow 0, H_L \leftarrow 0$
- 　　for $j$ in $\text{sorted}(I_k, \text{ascent order by } x_{jk})$ do
- 　　　　$G_L \leftarrow G_L + g_j, H_L \leftarrow H_L + h_j$
- 　　　　$G_R \leftarrow G - G_L, H_L \leftarrow H - H_L$
- 　　　　$\text{score} \leftarrow \max\{\text{score}, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda}\}$
- 　　end
- 　　// enumerate missing value goto left
- 　　$G_R \leftarrow 0, H_R \leftarrow 0$
- 　　for $j$ in $\text{sorted}(I_k, \text{ascent order by } x_{jk})$ do
- 　　　　$G_R \leftarrow G_R + g_j, H_R \leftarrow H_R + h_j$
- 　　　　$G_L \leftarrow G - G_R, H_L \leftarrow H - H_R$
- 　　　　$\text{score} \leftarrow \max\{\text{score}, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda}\}$
- 　　end
- end
- 输出：Split and default directions with max gain

上述的算法只遍历非缺失值。划分的方向怎么学呢？很naive但是很有效的方法：

- 让特征$k$的所有缺失值的都到右子树，然后和之前的一样，枚举划分点，计算最大的gain
- 让特征$k$的所有缺失值的都到左子树，然后和之前的一样，枚举划分点，计算最大的gain
- 这样最后求出最大增益的同时，也知道了缺失值的样本应该往左边还是往右边。

# 4.3. 系统设计

## 4.3.1. 分块并行 Column Block for Parallel Learning

在建树的过程中，最耗时是找最优的切分点，而这个过程中，最耗时的部分是将数据排序。为了减少排序的时间，提出Block结构存储数据。

- Block中的数据以稀疏格式CSC进行存储
- Block中的特征进行排序（不对缺失值排序）
- Block 中特征还需存储指向样本的索引，这样才能根据特征的值来取梯度
- 一个Block中存储一个或多个特征的值

只需在建树前排序一次，后面节点分裂时可以直接根据索引得到梯度信息。

**时间复杂度分析：**

- 设树的最大深度为$d$，共有$K$棵树，$\|x\|_0$表示训练集中非缺失的数据量，$q$表示数据集中候选变量的个数(q通常为32～100)，$B$表示每个block中最大的行数
- **Exact greedy algorithm**：original sparse aware algorithm $O(Kd\|x\|_0 \log n)$；on the block structure $O(Kd\|x\|_0 + \|x\|_0 \log n)$；基于Block的Exact greedy算法就省去了每一步中的排序开销。其中，$O(\|x\|_0 \log n)$ is the one time preprocessing cost that can be amortized
- **approximate algorithm**：original algorithm with binary search $O(Kd\|x\|_0 \log q)$；using the block structure $O(Kd\|x\|_0 + \|x\|_0 \log B)$

不同的Block可以在不同的机器上计算。该方法对Local策略尤其有效，因为Local策略每次分支都重新生成候选切分点。
Block结构还有其它好处，数据按列存储，可以同时访问所有的列，很容易实现并行的寻找分裂点算法。此外也可以方便实现之后要讲的out-of score计算。
缺点是空间消耗大了一倍。

### 4.3.2. 缓存优化 Cache-aware Access

### 4.3.3. Blocks for Out-of-core Computation

# 5. LightGBM

# 6. CatBoost

# 7. 参考链接

[XGBoost] https://www.hrwhisper.me/machine-learning-xgboost/