R 机器学习

第 12 讲　AdaBoost & GBDT

张敬信

2022 年 10 月 23 日

哈尔滨商业大学

将许多决策树作为弱学习器，按提升法做集成学习，产生了一系列强大的机器学习算法：Adaboost，GBDT，XGBoost，LightGBM，catBoost 等。

Adaboost，GBDT 是早期的提升树算法，XGBoost，LightGBM，catBoost 是近年出现的更复杂、性能更好的提升树算法。前者速度、性能都相对弱于后者，但学习其算法原理有助于理解后者。

一. AdaBoost

AdaBoost（Adaptive Boosting, 自适应提升），其自适应在于：前一个基分类器被错误分类的样本的权值会增大，而正确分类的样本的权值会减小，并再次用来训练下一个基分类器。同时，在每一轮迭代中，加入一个新的弱分类器，直到达到某个预定的足够小的错误率或达到预先指定的最大迭代次数。

后一个模型的训练永远是在前一个模型的基础上完成！

1. 算法思想

- 初始化训练样本的权值分布，每个样本具有相同权重；
- 训练弱分类器，如果样本分类正确，则在构造下一个训练集中，它的权值就会被降低；反之提高；用更新过的样本集去训练下一个分类器，一直这样迭代；
- 将所有弱分类器组合成强分类器，各个弱分类器的训练过程结束后，加大分类错误率小的弱分类器的权重，使其在最终的分类函数中起着较大的决定作用；降低分类错误率大的弱分类器的权重，使其在最终的分类函数中起着较小的决定作用。

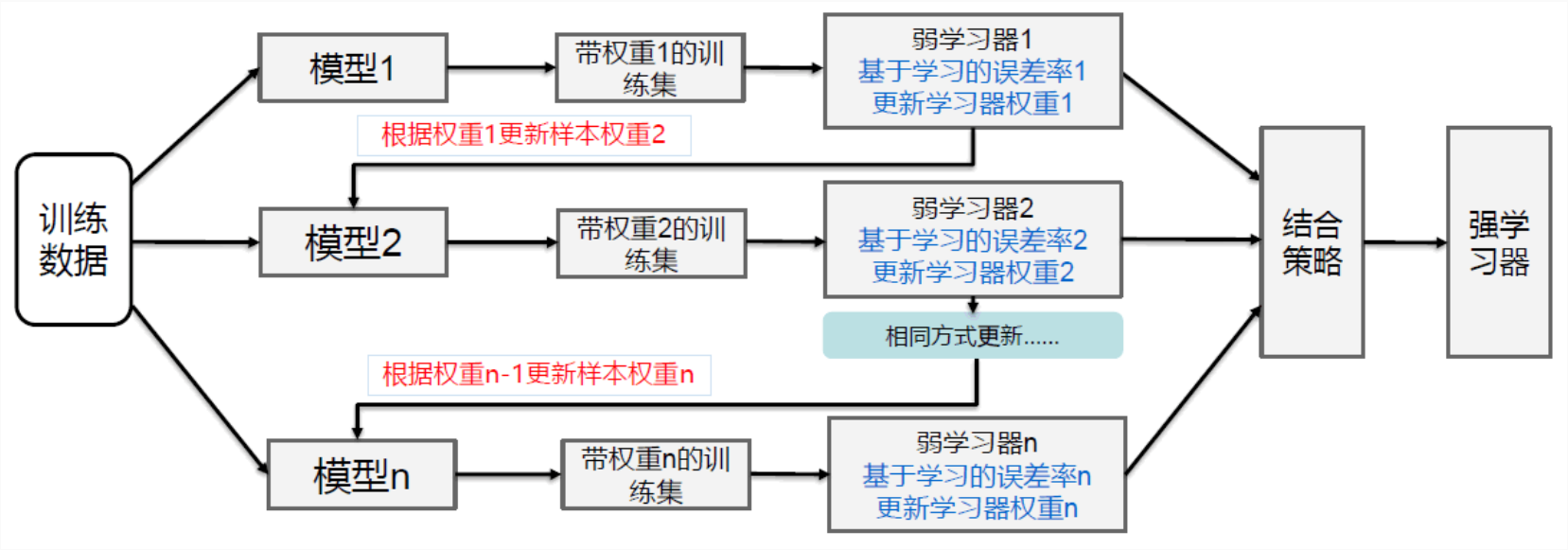


图 1: AdaBoost 算法示意图

2. 算法步骤

AdaBoost 是将多个弱学习器做线性加权组合成强学习器，第 M 轮的强学习器为：

$$f_M(x) = \sum_{m=1}^{M} \alpha_m G_m(x)$$

这里，权重

$$\alpha_m = \frac{1}{2} \ln \frac{1 - e_m}{e_m}$$

其中，

$$e_m = \frac{\sum_{i=1}^{n} w_i \mathbb{I}(y_i \neq G_m(x_i))}{\sum_{i=1}^{n} w_i}$$

表示第 m 个弱分类器的分类错误率。

该权重（推导略）可以保证弱分类器的分类错误率越低，相应的权重就越大。

AdaBoost 是采用指数损失：

$$\underset{\alpha, G}{\operatorname{argmin}} \sum_{i=1}^{n} \exp\left(-y_i f_M(x)\right)$$

在迭代训练过程中，AdaBoost 还要自适应修改样本的权重：分类正确的样本权重减小，分类错误的样本权重增大。

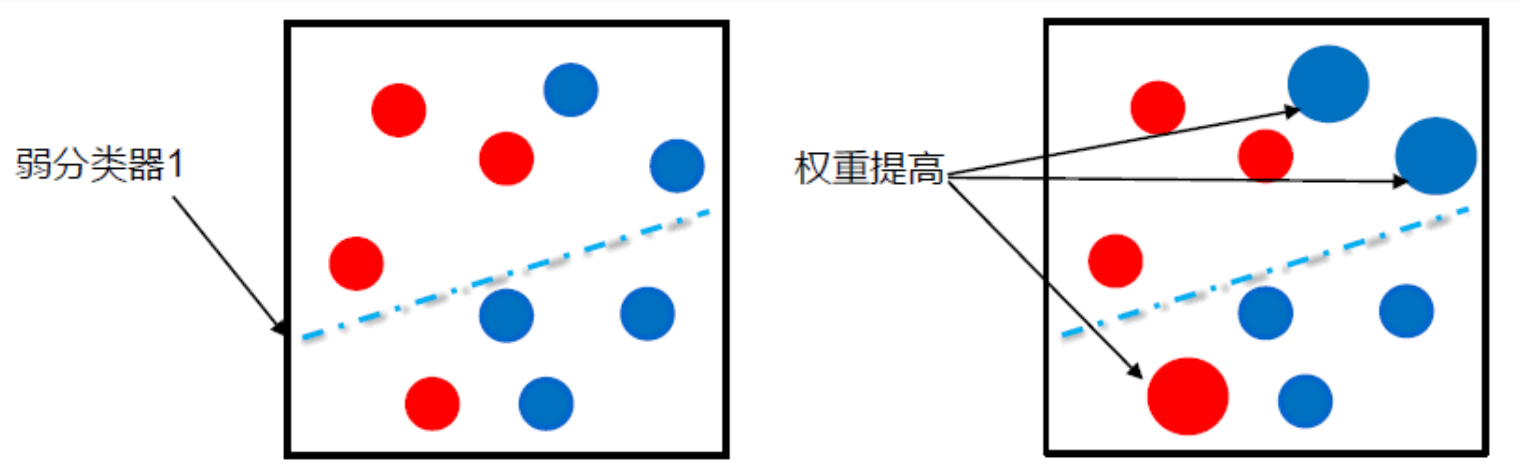


图 2: 样本权重更新示意图

各样本的初始权重均设为 $1/n$, 表示同等重要。第 $m+1$ 轮训练的样本权重集合:

$$D_{m+1} = (w_{m+1,1}, w_{m+1,2}, \cdots, w_{m+1,n})$$

是根据第 m 轮样本权重，以及第 m 轮分类器的准确率而定，计算公式为:

$$w_{m+1,i} = \frac{w_{m,i}}{Z_m} \exp\big(-\alpha_m y_i G_m(x_i)\big), \qquad i = 1, \cdots, n$$

其中，$Z_m = \sum_{i=1}^{n} w_{m,i} \exp\big(-\alpha_m y_i G_m(x_i)\big)$ 为归一化系数。

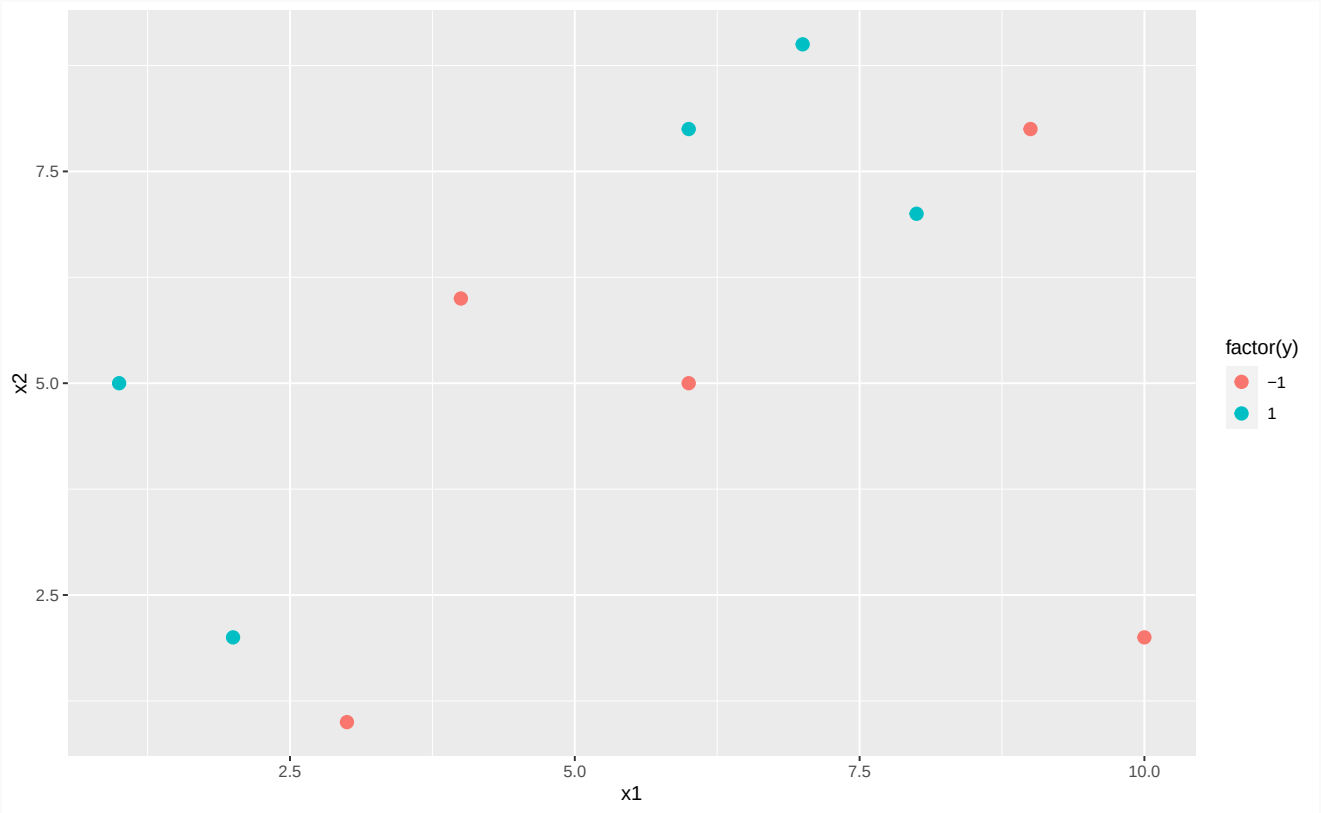
3. 简单算例

- 构造小数据集：

```
library(tidyverse)
df = tibble(x1 = c(1:4,6,6:10),
            x2 = c(5,2,1,6,8,5,9,7,8,2),
            y = c(1,1,-1,-1,1,-1,1,1,-1,-1))
head(df)
#> # A tibble: 6 x 3
#>      x1    x2     y
#>   <dbl> <dbl> <dbl>
#> 1     1     5     1
#> 2     2     2     1
#> 3     3     1    -1
#> 4     4     6    -1
#> 5     6     8     1
#> 6     6     5    -1
```

- 可视化

```
ggplot(df, aes(x1, x2, color = factor(y))) +
  geom_point(size = 3)
```



- 假设已有这样 3 个弱分类器

$$G_1(x) = \begin{cases} 1, & x_1 < 2.5 \\ -1, & x_1 > 2.5 \end{cases} \qquad G_2(x) = \begin{cases} 1, & x_1 < 8.5 \\ -1, & x_1 > 8.5 \end{cases}$$

$$G_3(x) = \begin{cases} 1, & x_2 > 6.5 \\ -1, & x_2 < 6.5 \end{cases}$$

```r
G1 = function(x) ifelse(x < 2.5, 1, -1)
G2 = function(x) ifelse(x < 8.5, 1, -1)
G3 = function(x) ifelse(x > 6.5, 1, -1)
train = df %>%                  # 生成 3 个分类器的预测结果
  mutate(yh1 = G1(x1), yh2 = G2(x1), yh3 = G3(x2))
```

第 1 轮迭代

- 初始化样本权重 D_1，计算 3 个分类器在其下的预测错误率

```
D1 = rep(1/10, 10)       # 初始化样本权重
err1 = train %>%
  summarise(across(contains("yh"), ~ sum((.x != y) * D1)))
err1
#> # A tibble: 1 x 3
#>     yh1   yh2   yh3
#>   <dbl> <dbl> <dbl>
#> 1   0.3   0.3   0.3
```

- 错误率都是 0.3, 选择 G_1 分类器，计算其权重：

```
e1 = err1$yh1
a1 = 0.5 * log((1 - e1) / e1)
```

- 构建强分类器：

$$f_1(x) = \alpha_1 G_1(x)$$

需要再接一个 `sign()` 到类别值。

```r
f1 = df %>%
  mutate(pre = sign(a1 * G1(x1)),
         rlt = ifelse(y == pre, TRUE, FALSE))
head(f1)                  # 预测结果, 样本 5,7,8 分错
#> # A tibble: 6 x 5
#>       x1    x2     y   pre rlt
#>    <dbl> <dbl> <dbl> <dbl> <lgl>
#> 1     1     5     1     1 TRUE
#> 2     2     2     1     1 TRUE
#> 3     3     1    -1    -1 TRUE
#> 4     4     6    -1    -1 TRUE
#> 5     6     8     1    -1 FALSE
#> 6     6     5    -1    -1 TRUE
1 - mean(f1$rlt)          # 预测错误率
#> [1] 0.3
```

第 2 轮迭代

- 更新权重到 D_2：样本 5,7,8 分错, 权重增大; 反之, 其它权重减小

```
t = exp(-a1 * train$y * train$yh1)
Z1 = sum(D1 * t)
D2 = D1 * t / Z1
D2
#>  [1] 0.0714 0.0714 0.0714 0.0714 0.1667 0.0714 0.1667 0.16
```

- 计算 3 个分类器在 D_2 下的预测错误率

```
err2 = train %>%
  summarise(across(contains("yh"), ~ sum((.x != y) * D2)))
err2
#> # A tibble: 1 x 3
#>     yh1   yh2   yh3
#>   <dbl> <dbl> <dbl>
#> 1   0.5 0.214 0.214
```

- G_2, G_3 分类器的错误率都是 0.214, 选择 G_2 分类器，计算其权重

```
e2 = err2$yh2
a2 = 0.5 * log((1 - e2) / e2)
```

- 构建强分类器：

$$f_2(x) = \alpha_1 G_1(x) + \alpha_2 G_2(x)$$

需要再接一个 `sign()` 到类别值。

```r
f2 = df %>%
  mutate(pre = sign(a1 * G1(x1) + a2 * G2(x1)),
         rlt = ifelse(y == pre, TRUE, FALSE))
head(f2)                  # 预测结果, 样本 3,4,6 分错
#> # A tibble: 6 x 5
#>       x1    x2     y   pre rlt
#>    <dbl> <dbl> <dbl> <dbl> <lgl>
#> 1     1     5     1     1 TRUE
#> 2     2     2     1     1 TRUE
#> 3     3     1    -1     1 FALSE
#> 4     4     6    -1     1 FALSE
#> 5     6     8     1     1 TRUE
#> 6     6     5    -1     1 FALSE
1 - mean(f2$rlt)          # 预测错误率
#> [1] 0.3
```

第 3 轮迭代

- 更新权重到 D_3：样本 3, 4, 6 分错, 权重增大; 反之, 其它权重减小

```
t = exp(-a2 * train$y * train$yh2)
Z2 = sum(D2 * t)
D3 = D2 * t / Z2
D3
#>  [1] 0.0455 0.0455 0.1667 0.1667 0.1061 0.1667 0.1061 0.10
```

- 计算 3 个分类器在 D_3 下的预测错误率

```
err3 = train %>%
  summarise(across(contains("yh"), ~ sum((.x != y) * D3)))
err3
#> # A tibble: 1 x 3
#>     yh1   yh2   yh3
#>   <dbl> <dbl> <dbl>
#> 1 0.318   0.5 0.136
```

- 选择错误率最小的 G_3 分类器，计算其权重

```
e3 = err3$yh3
a3 = 0.5 * log((1 - e3) / e3)
```

- 构建强分类器：

$$f_3(x) = \alpha_1 G_1(x) + \alpha_2 G_2(x) + \alpha_3 G_3(x)$$

需要再接一个 `sign()` 到类别值。

```r
f3 = df %>%
  mutate(pre = sign(a1 * G1(x1) + a2 * G2(x1) + a3 * G3(x2)),
         rlt = ifelse(y == pre, TRUE, FALSE))
head(f3, 3)                 # 预测结果
#> # A tibble: 3 x 5
#>       x1    x2     y   pre rlt
#>    <dbl> <dbl> <dbl> <dbl> <lgl>
#> 1     1     5     1     1 TRUE
#> 2     2     2     1     1 TRUE
#> 3     3     1    -1    -1 TRUE
1 - mean(f3$rlt)            # 预测错误率
#> [1] 0
```

经过 3 轮迭代，已经全部预测正确！

4. 其它

- AdaBoost 做多分类，主要区别是弱分类器的权重系数：

$$\alpha_m = \frac{1}{2}\ln\frac{1-e_m}{e_m} + \ln(K-1)$$

其中，K 为类别数。

- AdaBoost 回归，以 R2 回归为例，是将每个弱学习器的分类错误率换成回归误差率：

$$e_m = \sum_{i=1}^{n} w_{m,i} e_{mi}$$

其中，e_{mi} 为第 m 个学习器下每个样本的相对误差率（损失），可以是线性、平方、指数误差损失。

- 然后，弱学习器的权重为

$$\alpha_m = \frac{1 - e_m}{e_m}$$

- 再更新样本权重：

$$w_{m+1,i} = \frac{w_{mi}}{Z_m} \alpha_m^{1-e_{mi}}$$

其中，$Z_m = \sum_{i=1}^{n} w_{m,i} \alpha_m^{1-e_{mi}}$ 为规范化因子。

- 最后，构建强学习器

$$f_M(x) = \sum_{m=1}^{M} \ln(\frac{1}{\alpha_m}) g(x)$$

其中，$g(x)$ 为 $\alpha_m G_m(x),\ m = 1, \cdots, M$ 的中位数。

AdaBoost 优缺点

- 优点
  - 提供了一种框架，在其内可以用各种方法构建弱分类器，不用筛选特征，也不存在过拟合；
  - 不需要弱分类器的先验知识，最后得到的强分类器的分类精度依赖于所有弱分类器，能显著的提高学习精度；
  - 不需要预先知道弱分类器的错误率上限，可以根据弱分类器的反馈，自适应地调整假定的错误率，执行的效率高；
  - 根据训练集训练不同的弱分类器，用提升法集成起来构造一个分类能力很强的强分类器，即“三个臭皮匠赛过一个诸葛亮”。
- 缺点
  - 在训练过程中，难分类的样本权重呈指数增长，训练将会过于偏向该类样本，导致易受噪声干扰
  - 依赖于弱分类器，弱分类器的训练时间往往很长。

二. GBDT

- GBDT（Gradient Boosting Decison Tree, 梯度提升树），是由是由决策树、提升模型和梯度下降一起构成。GBDT 是公认的泛化能力较强的算法。
- AdaBoost 与 GBDT
  - AdaBoost 是通过提高错分样本的权重来定位模型的不足，采用指数损失，基分类器通常是为单层决策树；
  - GBDT 是通过负梯度来定位模型的不足，可以使用更多种类的损失函数

GBDT 中的决策树是回归树，因为每次迭代要拟合的是连续型梯度值，最适合用来做回归，调整后也可以用于分类。

通俗理解 GBDT：假设某人月薪 10k，先用一棵决策树拟合了 6k，发现有 4k 的损失（残差），然后再用一棵决策树拟合了残差中的 2k，这样持续拟合下去，拟合值和目标值之间的残差会越来越小，而将每一轮迭代，也就是每一棵决策树的预测值加起来就是模型最终的预测结果。

不停地用单棵决策树线性组合就是提升的过程，用梯度下降对提升树模型进行优化的过程就是梯度提升。

提升树模型可表示为：

$$f_M(x) = \sum_{m=1}^{M} T(x; \Theta_m)$$

给定初始模型为 $f_0(x) = 0$, 则第 m 步的提升树模型可表示为：

$$f_m(x) = f_{m-1}(x) + T(x; \Theta_m), \qquad m = 1, \cdots, M$$

对于回归提升树模型，一棵决策树可表示为：

$$T(x; \Theta) = \sum_{j=1}^{J} c_j \mathbb{I}(x \in R_j)$$

在给定第 $m-1$ 步的模型下，通过求解如下目标函数，来优化下一棵决策树的参数：

$$\hat{\Theta}_m = \underset{\Theta_m}{\operatorname{argmin}} \sum_{i=1}^{n} L(y_i,\ f_{m-1}(x_i) + T(x_i; \Theta_m))$$

当损失函数为平方误差损失时，

$$\begin{aligned} L(y, f_m(x)) = \left(y - f_m(x)\right)^2 &= [y - f_{m-1}(x) - T(x; \Theta_m)]^2 \\ &= [r - T(x; \Theta_m)]^2 \end{aligned}$$

其中，$r = y - f_{m-1}(x)$ 为残差。

所以，**第 m 棵决策树是对该残差的拟合！**

TABLE 10.2. *Gradients for commonly used loss functions.*

Setting	Loss Function	$-\partial L(y_i, f(x_i))/\partial f(x_i)$
Regression	$\frac{1}{2}[y_i - f(x_i)]^2$	$y_i - f(x_i)$
Regression	$\lvert y_i - f(x_i)\rvert$	$\text{sign}[y_i - f(x_i)]$
Regression	Huber	$y_i - f(x_i)$ for $\lvert y_i - f(x_i)\rvert \le \delta_m$ $\delta_m \text{sign}[y_i - f(x_i)]$ for $\lvert y_i - f(x_i)\rvert > \delta_m$ where $\delta_m = \alpha$th-quantile$\{\lvert y_i - f(x_i)\rvert\}$
Classification	Deviance	kth component: $I(y_i = \mathcal{G}_k) - p_k(x_i)$

图 3: GBDT 常用损失函数及负梯度损失

当损失函数为平方损失或指数损失函数时，上述优化很简单。但对于一般的损失函数，如绝对值损失和 Huber 损失，上述优化就没那么容易。为此，Freidman 提出了梯度提升算法：

利用最速下降的近似方法，用损失函数的负梯度在当前模型的值作为回归提升树中残差的近似值

即

$$r_{mi} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f(x)=f_{m-1}(x)}$$

于是，综合提升树和梯度提升，就得到 GBDT 模型一般步骤：

(1) 初始化弱学习器

$$f_0(x) = \underset{c}{\operatorname{argmin}} \sum_{i=1}^{n} L(y_i, c)$$

(2) 对 $m = 1, \cdots, M$ 依次迭代：

- 对每个样本 $i = 1, \cdots, n$, 计算负梯度 r_{mi} 作为近似残差；
- 将近似残差作为新的样本值，即将 (x_i, r_{mi}), $i = 1, \cdots, n$ 作为训练数据，拟合一棵新的回归树 $f_m(x)$, 其叶节点区域为 R_{mj}, $j = 1, \cdots, J$, 其中 J 为回归树叶节点的个数；

- 对叶节点区域计算最佳拟合值

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

- 更新强学习器

$$f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J} c_{mj} \mathbb{I}(x \in R_{mj})$$

(3) 输出 $\widehat{f}(x) = f_M(x).$

注：通常在弱学习器合成强学习器时还多乘以一个收缩系数（学习率），可以避免过拟合。

2. 简单算例

- 现有不同天气状况下，打高尔夫球人数的数据：

```
load("datas/GBDTdatas.rda")
head(df)
#> # A tibble: 6 x 6
#>     Day Outlook  Temp. Humidity Wind   Decision
#>   <int> <chr>    <chr> <chr>    <chr>     <int>
#> 1     1 Sunny    Hot   High     Weak         25
#> 2     2 Sunny    Hot   High     Strong       30
#> 3     3 Overcast Hot   High     Weak         46
#> 4     4 Rain     Mild  High     Weak         45
#> 5     5 Rain     Cool  Normal   Weak         52
#> 6     6 Rain     Cool  Normal   Strong       23
```

- 在平方损失下，GBDT 算法就是依次对残差训练回归树。对于回归树，是遍历各特征的值对因变量进行划分，分到同一组的样本预测值即为均值，划分的不纯度可以用标准差来衡量，所以划分依据是标准差增量最大。
- 先定义几个函数：

```
# 计算标准差
std = function(x) sqrt(sum((x - mean(x)) ^ 2) / length(x))
# 根据分类特征计算分组标准差
wstd = function(var, data) {
  data %>%
    group_by(.data[[var]]) %>%
    summarise(n = n(), std = std(Decision)) %>%
    summarise(WSTD = sum(n * std) / sum(n))}
```

注：若是连续特征，代码需要做相应修改。

```r
# 计算各特征的标准差增量
gain = function(df, y) {
  Vars = setdiff(names(df), y)
  Vars %>%
    map_dfr(wstd, data = df) %>%
    transmute(var = Vars, SDR = std(df[[y]]) - WSTD)
}
```

- 为了让模型简单和避免过拟合，本例将最小分支节点数取为 5，即分支下叶节点少于 5 将不再划分。

第 1 轮迭代

```
df1 = df[-1]                # 去掉 ID 列
gain(df1, "Decision")
#> # A tibble: 4 x 2
#>   var         SDR
#>   <chr>     <dbl>
#> 1 Outlook   1.66
#> 2 Temp.     0.480
#> 3 Humidity 0.272
#> 4 Wind      0.282
```

- 可见，Outlook 特征是最优的划分特征

- 根据 Outlook 做划分，过滤掉小于 5 的分支，然后继续计算各分支的标准差增量

```
df1 %>%
  group_nest(Outlook) %>%
  filter(map_lgl(data, ~ nrow(.x) >= 5)) %>%
  mutate(gains = map(data, ~ gain(.x, "Decision"))) %>%
  unnest(gains)
#> # A tibble: 6 x 4
#>   Outlook                data var         SDR
#>   <chr>   <list<tibble[,4]>> <chr>     <dbl>
#> 1 Rain              [5 x 4] Temp.    0.679
#> 2 Rain              [5 x 4] Humidity 0.371
#> 3 Rain              [5 x 4] Wind     7.62
#> 4 Sunny             [5 x 4] Temp.    4.18
#> 5 Sunny             [5 x 4] Humidity 3.33
#> 6 Sunny             [5 x 4] Wind     0.847
```

- 构建决策树，各组均值作为预测值

```
tree1 = df1 %>%
  mutate(leafs =
    case_when(Outlook == "Sunny" & Temp. == "Cool" ~ "L1",
              Outlook == "Sunny" & Temp. == "Hot"  ~ "L2",
              Outlook == "Sunny" & Temp. == "Mild" ~ "L3",
              Outlook == "Rain" & Wind == "Weak"   ~ "L4",
              Outlook == "Rain" & Wind == "Strong" ~ "L5",
              Outlook == "Overcast"  ~ "L6")) %>%
  group_by(leafs) %>%
  mutate(pred = mean(Decision)) %>%
  ungroup()
```

```
head(tree1)
#> # A tibble: 6 x 7
#>   Outlook  Temp. Humidity Wind   Decision leafs  pred
#>   <chr>    <chr> <chr>    <chr>     <int> <chr> <dbl>
#> 1 Sunny    Hot   High     Weak         25 L2     27.5
#> 2 Sunny    Hot   High     Strong       30 L2     27.5
#> 3 Overcast Hot   High     Weak         46 L6     46.2
#> 4 Rain     Mild  High     Weak         45 L4     47.7
#> 5 Rain     Cool  Normal   Weak         52 L4     47.7
#> 6 Rain     Cool  Normal   Strong       23 L5     26.5
```

第 2 轮迭代

- 计算残差，作为新的因变量

```
df2 = tree1 %>%
  mutate(Decision = Decision - pred) %>%
  select(1:5)
head(df2)
#> # A tibble: 6 x 5
#>   Outlook  Temp. Humidity Wind   Decision
#>   <chr>    <chr> <chr>    <chr>     <dbl>
#> 1 Sunny    Hot   High     Weak      -2.5
#> 2 Sunny    Hot   High     Strong     2.5
#> 3 Overcast Hot   High     Weak      -0.25
#> 4 Rain     Mild  High     Weak      -2.67
#> 5 Rain     Cool  Normal   Weak       4.33
#> 6 Rain     Cool  Normal   Strong    -3.5
```

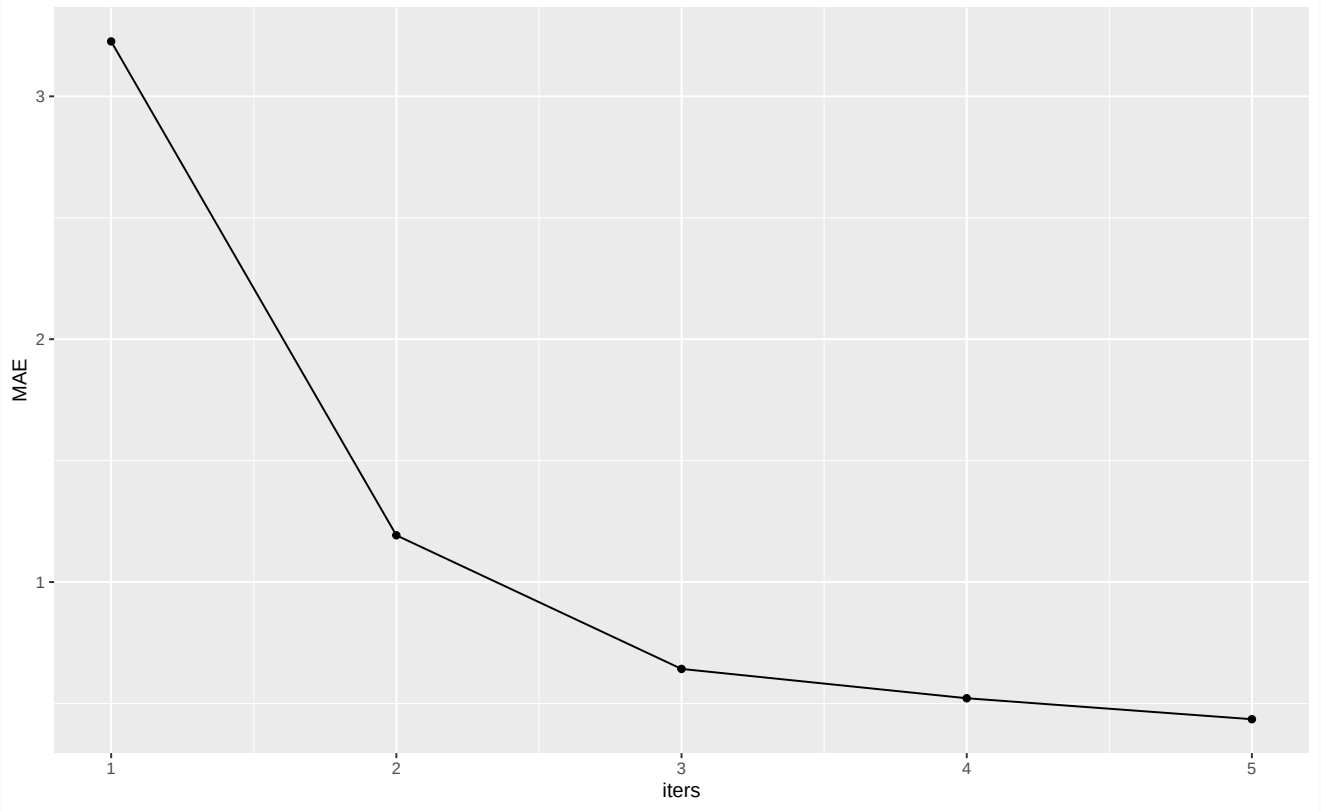
- 然后，同样地做第 3 轮迭代，第 4 轮迭代，第 5 轮迭代，得到汇总结果如下：

```
head(predictions)
#> # A tibble: 6 x 7
#>   Epoch1 Epoch2 Epoch3 Epoch4    Epoch5  pred actual
#>    <dbl>  <dbl>  <dbl>  <dbl>     <dbl> <dbl>  <int>
#> 1   27.5  -1.67 -1.96   0.152  0          24.0     25
#> 2   27.5   2.5   0.625  0.152  0          30.8     30
#> 3   46.2  -1.67  0.365  0.152  0          45.1     46
#> 4   47.7  -3.61  1.69  -0.586 -1.88e- 1  45.0     45
#> 5   47.7   2.17  1.69   0.213 -2.78e-17  51.7     52
#> 6   26.5  -3.38 -0.938  0.213 -2.78e-17  22.4     23
```

```
head(errors)
#> # A tibble: 6 x 5
#>    err1  err2  err3  err4   err5
#>   <dbl> <dbl> <dbl> <dbl>  <dbl>
#> 1  2.5  0.833 1.13  0.977 0.977
#> 2  2.5  0     0.625 0.777 0.777
#> 3  0.25 1.42  1.05  0.900 0.900
#> 4  2.67 0.944 0.741 0.155 0.0333
#> 5  4.33 2.17  0.481 0.269 0.269
#> 6  3.5  0.125 0.812 0.600 0.600
```

- 五轮迭代的 MAE（平均绝对误差）：

```
#> Warning: Unknown or uninitialised column: `y`.
#> NULL
```



GBDT 分类原理并无不同，只是损失函数要么采用指数损失，此时 GBDT 退化为 Adaboost；要么采用对数似然损失，类似二元或多元 Logistic 回归，用类别的预测概率值和真实概率值之差来拟合损失。

- 二元 GBDT 分类

采用对数似然损失：

$$L(y, f(x)) = \ln\left(1 + \exp(-yf(x))\right)$$

其中，$y \in \{-1, 1\}$. 此时的负梯度误差为：

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

对于生成的决策树，各个叶节点的最佳残差拟合值为：

$$c_{mj} = \underset{c}{\operatorname{argmin}} \sum_{x_i \in R_{mj}} \ln\left(1 + \exp(-y_i f_{m-1}(x_i) + c)\right)$$

由于上式比较难优化，一般用如下近似值代替：

$$c_{mj} = \sum_{x_i \in R_{mj}} r_{mi} \Big/ \sum_{x_i \in R_{mj}} |r_{mi}|(1 - |r_{mi}|)$$

多元 GBDT

采用对数似然损失：

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

若样本输出类别为 k, 则 $y_k = 1$. 第 k 类的概率为：

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

可计算第 m 轮第 i 个样本对应类别 l 的负梯度误差为：

$$r_{mil} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f(x)=f_{l,m-1}(x)} = y_{il} - p_{l,m-1}(x_i)$$

即样本 i 对应类别 l 的真实概率与 $m-1$ 轮预测概率之差。

对于生成的决策树，各个叶节点的最佳残差拟合值为：

$$c_{mjl} = \underset{c_{jl}}{\operatorname{argmin}} \sum_{i=1}^{n} \sum_{k=1}^{K} L(y_k, f_{m-1,l}(x) + \sum_{j=1}^{J} c_{jl} \mathbb{I}(x_i \in R_{mj}))$$

由于上式比较难优化，一般用如下近似值代替：

$$c_{mjl} = \frac{K-1}{K} \sum_{x_i \in R_{mjl}} r_{mil} \Big/ \sum_{x_i \in R_{mjl}} |r_{mil}|(1 - |r_{mil}|)$$

优缺点

- 优点：
  - 处理各种类型的数据（非线性数据），包括连续值和离散值，处理多特征类型；
  - 在相对少的调参时间情况下，预测的准备率也可以比较高（相对 SVM）；[1]
  - 能适应多种损失函数，使用健壮的损失函数，对异常值的鲁棒性强。如 Huber 损失函数和 Quantile 损失函数；
  - 适合低维稠密数据，模型可解释性好；
  - 不需要做特征的归一化，可以自动选择特征。

[1]调参范围: 树的棵数 100-10000、深度 3-8、学习速率 0.01-1、最大叶节点数 20、训练采样比例 0.5 1、训练特征采样比例 $\sqrt{n}/n$.

- 缺点：
  - 弱学习器之间相互依赖，难以并行训练数据。可以通过自采样 SGBT 达到部分并行；
  - 计算复杂度大；
  - 不使用高维稀疏特征。

主要参考文献

[1] Jim Liang(梁劲). Getting Started with Machine Learning, 2019.

[2] Miracle8070. 白话机器学习：算法理论 + 实战之 Ada Boost 算法.

[3] 刘建平. 集成学习之 Adaboost 算法原理小结. 博客园.

[4] 刘建平. 梯度提升树 (GBDT) 原理小结. 博客园.

[4] 黄海广. 机器学习课件全集. 温州大学, 2021.

[5] Sefik Ilkin Serengil. A Step by Step Gradient Boosting Decision Tree Example, 2018.