

R 机器学习

第 13 讲 XGBoost

张敬信

2022 年 10 月 23 日

哈尔滨商业大学

XGBoost (eXtremeGradient Boosting, 陈天奇) 是大规模并行提升树算法, 它是目前最快最好的开源提升树算法, 比传统提升树算法快 10 倍以上。Xgboost 和 GBDT 两者都是 Boosting 方法, 除了工程实现、解决问题上的一些差异外, 最大的不同就是目标函数的定义。

- GBDT 将目标函数泰勒展开到一阶，而 XGBoost 将目标函数泰勒展开到了二阶，保留了更多有关目标函数的信息，有助于提升效果。
- GBDT 是给新的基模型寻找新的拟合标签（前面加法模型的负梯度），而 XGBoost 是给新的基模型寻找新的目标函数（目标函数关于新的基模型的二阶泰勒展开）。
- XGBoost 加入了叶子权重的 L_2 正则化项，从而降低模型方差。
- XGBoost 增加了自动处理缺失值特征的策略。通过把带缺失值样本分别划分到左子树或者右子树，比较两种方案下目标函数的优劣，从而自动对有缺失值的样本进行划分，无需对缺失特征进行填充预处理。
- XGBoost 还支持候选分位点切割，特征并行等。

一. 算法原理

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

$$\hat{y}_i = \sum_{t=1}^k f_t(x_i)$$

其中， f_k 为第 k 个弱学习器， \hat{y}_i 为第 i 个样本的预测值。

损失函数可由 \hat{y}_i 与真实值 y_i 来表示：

$$L = \sum_{i=1}^n l(y_i, \hat{y}_i)$$

其中， n 为样本数。

1. 目标函数

模型的预测精度由模型的偏差和方差共同决定，损失函数代表了模型的偏差，想要方差小则需要简单的模型，所以目标函数由模型的损失函数 L 与抑制模型复杂度的正则项 Ω 组成：

$$Obj = \sum_{i=1}^n l(y_i, \hat{y}_i) + \sum_{t=1}^k \Omega(f_t)$$

注：XGBoost 的基模型可以是决策树也可以是线性模型。

Boosting 模型是前向加法，以第 t 步的模型为例，模型对第 i 个样本 x_i 的预测为：

$$\hat{y}_i^t = \hat{y}_i^{t-1} + f_t(x_i)$$

其中， \hat{y}_i^{t-1} 为第 $t-1$ 步的预测值，是已知常数， $f_t(x_i)$ 是这次需要加入的新模型的预测值。

于是，目标函数可表示为：

$$\begin{aligned} Obj^{(t)} &= \sum_{i=1}^n l(y_i, \hat{y}_i^t) + \sum_{i=1}^t \Omega(f_i) \\ &= \sum_{i=1}^n l(y_i, \hat{y}_i^{t-1} + f_t(x_i)) + \sum_{i=1}^t \Omega(f_i) \end{aligned}$$

求此时最优化目标函数，就相当于求解 $f_t(x_i)$. 根据泰勒公式二阶展开式：

$$f(x + \Delta x) \approx f(x) + f'(x)\Delta x + \frac{1}{2}f''(x)\Delta x^2$$

(1) 损失函数简化

将 \hat{y}_i^{t-1} 看作 x , $f_t(x_i)$ 看作 Δx , 则

$$Obj^{(t)} = \sum_{i=1}^n [l(y_i, \hat{y}_i^{t-1}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \sum_{i=1}^t \Omega(f_i)$$

其中, g_i 为损失函数的一阶导, h_i 为损失函数的二阶导, 注意这里是指对 \hat{y}_i^{t-1} 求导。

以平方损失为例, 则

$$g_i = \frac{\partial (y_i - \hat{y}_i^{t-1})^2}{\partial \hat{y}_i^{t-1}} = -2(y_i - \hat{y}_i^{t-1})$$

$$h_i = \frac{\partial (-2(y_i - \hat{y}_i^{t-1}))}{\partial \hat{y}_i^{t-1}} = 2$$

由于在第 t 步时 \hat{y}_i^{t-1} 其实是一个已知的值，所以 $l(y_i, \hat{y}_i^{t-1})$ 是常数，对函数的优化不会产生影响，因此目标函数可以写为：

$$Obj^{(t)} \approx \sum_{i=1}^n [g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \sum_{i=1}^t \Omega(f_i)$$

所以，只要求出每一步损失函数的一阶导和二阶导的值（由于前一步的 \hat{y}^{t-1} 是已知的，故这两个值是常数），然后最优化目标函数，就可以得到每一步的 $f_t(x)$ ，最后根据加法模型得到一个整体模型。

(2) 正则项简化

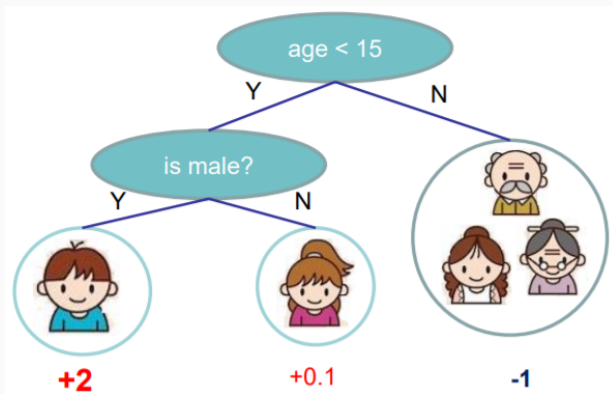
Xgboost 的基模型不仅支持决策树，还支持线性模型，这里只介绍基于决策树的目标函数。

将决策树记为 $f_t(x) = w_{q(x)}$, 其中, $q(x)$ 表示样本 x 在哪个叶节点, $w_{q(x)}$ 表示叶节点取值 (预测值) 为 w .

决策树的复杂度取决于叶节点数 T , 叶节点越少模型越简单, 此外, 叶节点也不应该含有过高的权值 w . 故目标函数的正则项可定义为:

$$\Omega(f_t) = \gamma T + \frac{1}{2} \lambda \sum_{j=1}^T w_j^2$$

即决策树模型的复杂度由叶节点数量和所有叶节点权值向量的 L_2 范数共同决定。



以该决策树为例, $\Omega = \gamma \cdot 3 + \frac{1}{2}\lambda(2^2 + 0.1^2 + (-1)^2)$.

记 $I_j = \{i : q(x_i) = j\}$ 为划分到第 j 个叶节点的样本集合, 则

$$\begin{aligned}
 Obj^{(t)} &\approx \sum_{i=1}^n [g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \sum_{i=1}^t \Omega(f_i) \\
 &= \sum_{i=1}^n \left[g_i w_{q(x_i)} + \frac{1}{2} h_i w_{q(x_i)}^2 \right] + \gamma T + \frac{1}{2} \lambda \sum_{j=1}^T w_j^2 \\
 &= \sum_{j=1}^T \left[\left(\sum_{i \in I_j} g_i \right) w_j + \frac{1}{2} \left(\sum_{i \in I_j} h_i + \lambda \right) w_j^2 \right] + \gamma T
 \end{aligned}$$

第二步是遍历所有样本后求每个样本的损失函数, 因为样本会最终落在叶节点上, 所以也可以遍历叶节点, 得到叶节点上的样本集合, 再求损失函数。即遍历对象从原来的样本集合改写为叶节点集合, 由于一个叶节点可能有多个样本, 所以是 $\sum_{i \in I_j} g_i$ 和 $\sum_{i \in I_j} h_i$ 。

为了简单, 分别记为 G_j 和 H_j , w_j 为第 j 个叶节点预测值, 则目标函数为:

$$Obj^{(t)} \approx \sum_{j=1}^T \left[G_j w_j + \frac{1}{2} (H_j + \lambda) w_j^2 \right] + \gamma T$$






注意, G_j 和 H_j 是前 $t - 1$ 步得到的结果, 其值已知可看作常数, 只有最后一棵数的叶节点 w_j 不确定, 那么将目标函数对 w_j 求一阶导, 并令其等于 0, 则可求得叶节点 j 对应的权值:

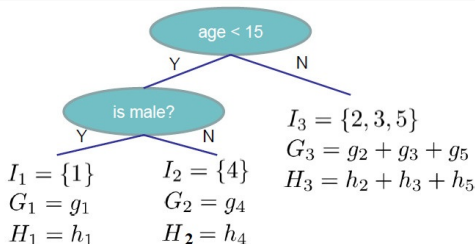
$$w_j^* = \frac{G_j}{H_j + \lambda}$$

所以, 目标函数最终可化简为:

$$Obj = -\frac{1}{2} \sum_{j=1}^T \frac{G_j^2}{H_j + \lambda} + \gamma T$$

样本号 梯度数据

1		g_1, h_1
2		g_2, h_2
3		g_3, h_3
4		g_4, h_4
5		g_5, h_5



$$Obj = - \sum_j \frac{G_j^2}{H_j + \lambda} + 3\gamma$$

该得分越小，表明树的结构越好

先求出每个节点每个样本的一阶导数 g_i 和二阶导数 h_i ，然后针对每个节点对所含样本求和得到 G_j 和 H_j ，最后遍历决策树的节点即可得到目标函数。

通过引入二阶导信息，XGBoost 的优化已经极为逼近真实损失，其节点分裂方式与 CART 树本质上是一样的，只是信息增益的计算方式有所不同。

假设模型在某一节点完成特征分裂，则分裂前的目标函数可以写为：

$$Obj_1 = -\frac{1}{2} \left[\frac{(G_L + G_R)^2}{H_L + H_R + \lambda} \right] + \gamma$$

分裂后的目标函数为：

$$Obj_1 = -\frac{1}{2} \left[\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} \right] + 2\gamma$$

则对于目标函数来说，分裂后的增益为：

$$Gain = \frac{1}{2} \left[\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda} \right] - \gamma$$

注：该特征增益也可以作为特征重要性的度量指标。

实际处理时，需要遍历所有特征寻找最佳分裂特征以及最优切分点。若增益 $Gain > 0$ ，即分裂为两个叶子节点后，目标函数下降了。

但还要考虑引入新叶节点的惩罚项，若分割带来的增益大于某阈值，则执行该分裂，否则可以剪枝掉该分割。

另外，XGBoost 模型在工程实现上还有一些有助于提速的技术细节：

- 最优切分点的分位数近似算法
- 加权分位数缩略图
- 稀疏感知算法
- 块结构设计
- 缓存访问优化算法
- “核外”块计算

优点:

- 精度更高: GBDT 只用到一阶泰勒展开, 而 XGBoost 对损失函数进行了二阶泰勒展开。XGBoost 引入二阶导一方面是为了增加精度, 另一方面也是为了能够自定义损失函数, 二阶泰勒展开可以近似大量损失函数;
- 灵活性更强: GBDT 以 CART 作为基分类器, XGBoost 不仅支持 CART 还支持线性分类器, (使用线性分类器的 XGBoost 相当于带 L_1 和 L_2 正则化项的 Logistic 回归或线性回归。此外, XGBoost 工具支持自定义损失函数, 只需函数支持一阶和二阶求导;
- 正则化: XGBoost 在目标函数中加入了正则项, 用于控制模型的复杂度。正则项里包含了树的叶子节点个数、叶子节点权重的 L_2 范数。正则项降低了模型的方差, 使学习出来的模型更加简单, 有助于防止过拟合;

- Shrinkage (缩减): 相当于学习率。XGBoost 在进行完一次迭代后, 会将叶子节点的权重乘上该系数, 主要是为了削弱每棵树的影响, 让后面有更大的学习空间;
- 列抽样: XGBoost 借鉴了随机森林的做法, 支持列抽样, 不仅能降低过拟合, 还能减少计算量;
- 缺失值处理: XGBoost 采用的稀疏感知算法极大的加快了节点分裂的速度; 可以并行化操作: 块结构可以很好的支持并行计算。

缺点:

- 虽然利用预排序和近似算法可以降低寻找最佳分裂点的计算量, 但在节点分裂过程中仍需要遍历数据集;
- 预排序过程的空间复杂度过高, 不仅需要存储特征值, 还需要存储特征对应样本的梯度统计值的索引, 相当于消耗了两倍的内存。

二. XGBoost 回归案例

- 以预测国王郡房价预测为例, `kc_housing` 数据集来自 `mlr3data` 包, 包含 21613 个样本, 20 个变量, 其中目标变量是 `price`, 特征包括房屋面积、条件、楼层, 经纬度、州代码等。
- 有些特征, 比如日期、是否翻新、州代码等需要做简单的预处理。
- 本例将阐述用 `mlr3verse` 做 XGBoost 回归的一般流程。

1. 准备数据

- 先加载包，载入数据集

```
library(mlr3verse)
library(tidyverse)
library(lubridate)
data("kc_housing", package = "mlr3data")
str(kc_housing)
#> 'data.frame':    21613 obs. of  20 variables:
#> $ date          : POSIXct, format: "2014-10-13" "2014-12-0
#> $ price         : num  221900 538000 180000 604000 510000
#> $ bedrooms      : int   3 3 2 4 3 4 3 3 3 3 ...
#> $ bathrooms     : num   1 2.25 1 3 2 4.5 2.25 1.5 1 2.5 ...
#> $ sqft_living   : int  1180 2570 770 1960 1680 5420 1715 1
#> $ sqft_lot      : int   5650 7242 10000 5000 8080 101930 68
#> $ floors        : num   1 2 1 1 1 1 2 1 1 2 ...
#> $ waterfront    : logi   FALSE FALSE FALSE FALSE FALSE FAIS
```

- XGBoost 学习器不支持日期特征、因子特征、字符串特征，需要做一些简单的特征预处理：
 - 将日期 `date` 变成距离最小日期的天数
 - 将 `yr_renovated` 根据其是否缺失定义为是否翻新
 - 将 `sqft_basement` 根据其是否缺失定义为是否有地下室
 - 将房价 `price` 除以 1000, 变成千美元
 - 将州代码 `zipcode` 做目标编码：赋值为该州的房屋均价

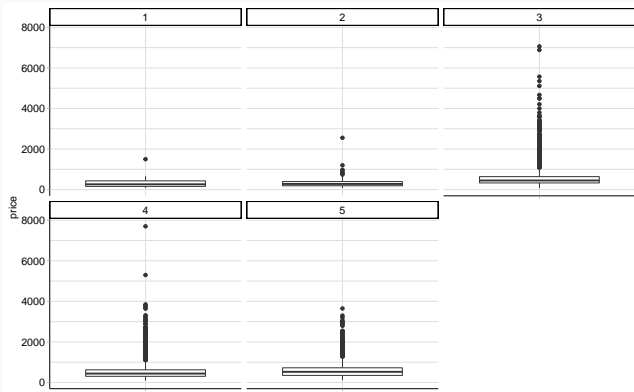
```
kc_housing = kc_housing %>%  
  mutate(date = min(date) %--% date / ddays(1),  
         yr_renovated = ifelse(is.na(yr_renovated), 0, 1),  
         sqft_basement = ifelse(is.na(sqft_basement), 0, 1),  
         price = price / 1000) %>%  
  group_by(zipcode) %>%  
  mutate(zipcode = mean(price)) %>%  
  ungroup()
```

2. 创建任务

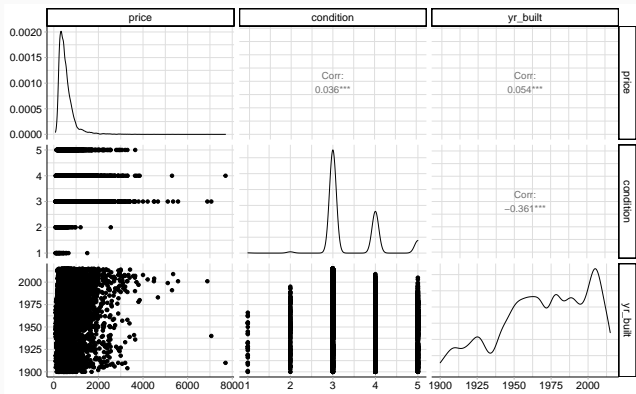
```
task = as_task_regr(kc_housing, target = "price")
task
#> <TaskRegr:kc_housing> (21613 x 20)
#> * Target: price
#> * Properties: -
#> * Features (19):
#>   - int (10): bedrooms, condition, grade, sqft_above, sqft
#>     sqft_living15, sqft_lot, sqft_lot15, view, yr_built
#>   - dbl (8): bathrooms, date, floors, lat, long, sqft_base
#>     yr_renovated, zipcode
#>   - lgl (1): waterfront
```

- 简单可视化探索

```
autoplot(task) +  
  facet_wrap(~ condition)
```




```
autoplot(task$clone())$select(task$feature_names[c(3, 17)]),  
  type = "pairs")
```



3. 选择学习器

选择学习器

```
lrn_xgb = lrn("regr.xgboost") # 需要 xgboost 包
```

```
lrn_xgb
```

```
#> <LearnerRegrXgboost:regr.xgboost>
```

```
#> * Model: -
```

```
#> * Parameters: nrounds=1, nthread=1, verbose=0
```

```
#> * Packages: mlr3, mlr3learners, xgboost
```

```
#> * Predict Types: [response]
```

```
#> * Feature Types: logical, integer, numeric
```

```
#> * Properties: hotstart_forward, importance, missings, weights
```

4. 划分训练集测试集

- 做留出 (holdout) 重抽样, 70% 作为训练集, 其余 30% 作为测试集
- 为了保持训练集、测试集的因变量数据具有相似的分布, 采用分层抽样方法
- 用 `partition()` 函数对任务做划分, 默认按因变量分层, 取出训练集索引和测试集索引

```
set.seed(123)
split = partition(task, ratio = 0.7)
# 默认 stratify = TRUE
```

5. 超参数调参

-查看学习器的超参数集

```
lrn_xgb$param_set
```

```
#> <ParamSet>
```

```
#>           id      class lower upper nleve
```

```
#> 1:           alpha ParamDbl      0   Inf      I
```

```
#> 2:      approxcontrib ParamLgl    NA    NA
```

```
#> 3:      base_score ParamDbl -Inf   Inf      I
```

```
#> 4:      booster ParamFct    NA    NA
```

```
#> 5:      callbacks ParamUty    NA    NA      I
```

```
#> 6: colsample_bylevel ParamDbl      0      1      I
```

```
#> 7: colsample_bynode ParamDbl      0      1      I
```

```
#> 8: colsample_bytree ParamDbl      0      1      I
```

```
#> 9: disable_default_eval_metric ParamLgl    NA    NA
```

```
#> 10: early_stopping_rounds ParamInt      1   Inf      I
```

```
#> 11:           eta ParamDbl      0      1
```

- 设置搜索空间，借用文献 [6] 设置

```
search_space = ps(  
    # 缩减（学习）率  
    eta = p_dbl(lower = 0.2, upper = 0.4),  
    # 最小叶节点权重和  
    min_child_weight = p_dbl(lower = 1, upper = 20),  
    # 子采样比例  
    subsample = p_dbl(lower = 0.7, upper = 0.8),  
    # 树的列采样比例  
    colsample_bytree = p_dbl(lower = 0.9, upper = 1),  
    # 层的列采样比例  
    colsample_bylevel = p_dbl(lower = 0.5, upper = 0.7),  
    # 最大提升代数  
    nrounds = p_int(lower = 1, upper = 25))
```

- 设置自动调参器

```
at = auto_tuner(  
    method = "random_search",  
    learner = lrn_xgb,  
    resampling = rsmp("holdout"),  
    measure = msr("regr.rmse"),  
    search_space = search_space,  
    term_evals = 10,  
    batch_size = 40)
```

- 在训练集上启动调参过程，并启用并行化加速

```
future::plan("multicore")    # 启动并行化
```

```
set.seed(1)
```

```
at$train(task, row_ids = split$train)
```

```
#> INFO [13:56:02.197] [bbotk] Starting to optimize 6 parameters
```

```
#> INFO [13:56:02.252] [bbotk] Evaluating 40 configurations
```

```
#> INFO [13:56:04.052] [mlr3] Running benchmark with 40 resamples
```

```
#> INFO [13:56:04.079] [mlr3] Applying learner 'regr.xgboost'
```

```
#> INFO [13:56:04.300] [mlr3] Applying learner 'regr.xgboost'
```

```
#> INFO [13:56:04.412] [mlr3] Applying learner 'regr.xgboost'
```

```
#> INFO [13:56:04.677] [mlr3] Applying learner 'regr.xgboost'
```

```
#> INFO [13:56:04.861] [mlr3] Applying learner 'regr.xgboost'
```

```
#> INFO [13:56:04.912] [mlr3] Applying learner 'regr.xgboost'
```

```
#> INFO [13:56:05.159] [mlr3] Applying learner 'regr.xgboost'
```

```
#> INFO [13:56:05.319] [mlr3] Applying learner 'regr.xgboost'
```

```
#> INFO [13:56:05.511] [mlr3] Applying learner 'regr.xgboost'
```

```
#> INFO [13:56:05.762] [mlr3] Applying learner 'regr.xgboost'
```

- 查看最优超参数

```
at$tuning_result
```

```
#>      eta min_child_weight subsample colsample_bytree colsa  
#> 1: 0.247          3.39      0.773          0.905  
#> learner_param_vals  x_domain regr.rmse  
#> 1:          <list[8]> <list[6]>      124
```


- 用调出的最优参数更新学习器的参数集，然后训练模型

```
lrn_xgb$param_set$values =  
  at$tuning_result$learner_param_vals[[1]]  
lrn_xgb$train(task, row_ids = split$train)
```

6. 模型预测与评估

```
predictions = lrn_xgb$predict(task, row_ids = split$test)
predictions$score(msr("regr.rmse"))
#> regr.rmse
#>          126
```

7. 预测新数据

```
newdata = kc_housing[1:5, -2]
lrn_xgb$predict_newdata(newdata)
#> <PredictionRegr> for 5 observations:
#>   row_ids truth response
#>       1    NA      220
#>       2    NA      513
#>       3    NA      349
#>       4    NA      471
#>       5    NA      489
```

- [1] mlr3book. 2021. <https://mlr3book.mlr-org.com/>
- [2] 阿泽. 决策树（下）——XGBoost、LightGBM（非常详细）, 2020.
- [3] 陈天奇. Introduction to Boosted Trees (PPT). 2014
- [4] Miracle8070. 白话机器学习：算法理论 + 实战之 XGBoost 算法.
- [5] louwill. 数学推导 + 纯 Python 实现机器学习算法 17: XGBoost, 2020.
- [6] Florian Pfisterer. mlr3gallery: House Prices in King County.
<https://mlr3gallery.mlr-org.com/posts/2020-01-30-house-prices-in-king-county/>