集成学习之XGBoost & LightGBM

Boosting

- 个体学习器之间存在强依赖关系、训练时必须是串行生成的序列化方法。

GBDT (Gradient Boosting Decision Tree, Boosting 的一种)

- 由多棵回归树组成的森林，后一棵树以前面树林的结果与真实结果的残差为拟合目标
- 最终通过分类器组合加权的方式，进一步减小误差

$$C(x) = \sum_{m=1}^{M} w_m C_m(x)$$

XGBoost: A Scalable Tree Boosting System, 2016

XGBoost 是陈天奇等人开发的一个开源机器学习项目，高效地实现了 GBDT 算法并进行了算法和工程上的许多改进。

算法层面

损失函数分析

$$\begin{aligned}
\mathcal{L}^{(t)} &= \sum_{i=1}^{n} l(y_i, \hat{y}^{(t-1)} + C_t(x_i)) + \gamma T + \frac{1}{2}\lambda\|w\|^2 \\
\text{泰勒二阶展开} &\backsimeq \sum_{i=1}^{n} [l(y_i, \hat{y}^{(t-1)}) + g_i C_t(x_i) + \frac{1}{2} h_i C_t^2(x_i)] + \gamma T + \frac{1}{2}\lambda\|w\|^2 \\
\text{去除常数项得} \tilde{\mathcal{L}}^{(t)} &= \sum_{i=1}^{n} [g_i C_t(x_i) + \frac{1}{2} h_i C_t^2(x_i)] + \gamma T + \frac{1}{2}\lambda\|w\|^2 \\
\text{决策树只有只有叶子节点才有最终样本分布，以叶节点}j\text{为例，}I_j\text{为该节点上的分布样本} & \\
&= \sum_{j=1}^{T} [(\sum_{i\in I_j} g_i) w_j + \frac{1}{2}(\sum_{i\in I_j} h_i + \lambda) w_j^2] + \gamma T \\
\text{损失函数二元一次方程最值点在} w_j^* &= -\frac{\sum_{i\in I_j} g_i}{\sum_{i\in I_j} h_i + \lambda} \text{取得，即} \\
\tilde{\mathcal{L}}^{(t)}(q) &= -\frac{1}{2}\sum_{j=1}^{T} \frac{(\sum_{i\in I_j} g_i)^2}{\sum_{i\in I_j} h_i + \lambda} + \gamma T
\end{aligned} \tag{1}$$

> t : 迭代轮次数; n : 样本数; T : 树节点数; q : 树结构
>
> g_i : 一阶导数; h_i : 二阶导数。展开到二阶的原因：损失函数的一元二次方程最大值容易求得
>
> 蓝色字体为正则化项

候选点划分信息增益指标

$$\begin{aligned}
\mathcal{L}_{split} &= \mathcal{L}_{after_split} - \mathcal{L}_{before_split} \\
&= \frac{1}{2}\left[\frac{(\sum_{i\in I_L} g_i)^2}{\sum_{i\in I_L} h_i + \lambda} + \frac{(\sum_{i\in I_R} g_i)^2}{\sum_{i\in I_R} h_i + \lambda} - \frac{(\sum_{i\in I} g_i)^2}{\sum_{i\in I} h_i + \lambda}\right] + \gamma
\end{aligned} \tag{2}$$

split findling: pre-sorted algorithm

遍历每个划分点进行划分

Algorithm 1: Exact Greedy Algorithm for Split Finding

Input: I, instance set of current node
Input: 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**
  $G_L \leftarrow 0,\ H_L \leftarrow 0$
  for j *in sorted*$(I,$ *by* $\mathbf{x}_{jk})$ **do**
    $G_L \leftarrow G_L + g_j,\ H_L \leftarrow H_L + h_j$
    $G_R \leftarrow G - G_L,\ H_R \leftarrow H - H_L$
    $score \leftarrow \max(score, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda})$
  end
end
Output: Split with max score

1. 对于每个特征，依据特征值对每个样本进行升序排序
2. 遍历每个特征值，选择最大信息增益值对应的划分点

> 排序加遍历复杂度： $O(\#data * \log \#data + \#data * \#feature)$

split findling: histogram-based algorithm

将耗时的遍历算法转化为分位数近似算法，寻找划分点

Algorithm 2: Approximate Algorithm for Split Finding

for $k = 1$ ***to*** m **do**
  Propose $S_k = \{s_{k1}, s_{k2}, \cdots s_{kl}\}$ by percentiles on feature k.
  Proposal can be done per tree (global), or per split(local).
end
for $k = 1$ ***to*** m **do**
  $G_{kv} \leftarrow= \sum_{j \in \{j | s_{k,v} \geq \mathbf{x}_{jk} > s_{k,v-1}\}} g_j$
  $H_{kv} \leftarrow= \sum_{j \in \{j | s_{k,v} \geq \mathbf{x}_{jk} > s_{k,v-1}\}} h_j$
end
Follow same step as in previous section to find max score only among proposed splits.

1. 对于每个特征，根据特征值对每个样本进行升序排序，并计算截至每个特征值的分位数，即

$$r_k(z) = \frac{\sum_{(x,h) \in D_k, x<z} h}{\sum_{(x,h) \in D_k} h} \tag{3}$$

> 选择 h 计算分位数 (1) 式中单位二元一次方程的系数为 h，因此可对应于权重

2. 将样本放入等分的直方图中 $\{s_{k1}, s_{k2}, \ldots, s_{kl}\}, s_{k1} = \min_{i \in [1,n]} x_{ik}, s_{kl} = \max_{i \in [1,n]} x_{ik}$ ，要求

$$|r_k(s_{k,j}) - r_k(s_{k,j+1})| < \epsilon$$

3. 遍历直方图每个 bin 边界，选择最大信息增益值对应的划分点

> 排序加遍历复杂度： $O(\#data * \log \#data + \#bin * \#feature)$

sparsity-aware split algorithm

训练过程中输入特征稀疏，存在很多 0 值，造成稀疏性原因有

1. 在统计时词的“重要性”较低，最终被舍弃

2. 特征表示使用了 one-hot 编码

该算法考虑到了实际过程中特征的稀疏性，为了提升计算效率

Algorithm 3: Sparsity-aware Split Finding

Input: I, instance set of current node
Input: $I_k = \{i \in I | x_{ik} \neq \text{missing}\}$
Input: d, 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 sorted*$(I_k,$ *ascent order by* $\mathbf{x}_{jk})$ **do**
    $G_L \leftarrow G_L + g_j,\ H_L \leftarrow H_L + h_j$
    $G_R \leftarrow G - G_L,\ H_R \leftarrow H - H_L$
    $score \leftarrow \max(score, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda})$
  end
  // enumerate missing value goto left
  $G_R \leftarrow 0,\ H_R \leftarrow 0$
  for j *in sorted*$(I_k,$ *descent order by* $\mathbf{x}_{jk})$ **do**
    $G_R \leftarrow G_R + g_j,\ H_R \leftarrow H_R + h_j$
    $G_L \leftarrow G - G_R,\ H_L \leftarrow H - H_R$
    $score \leftarrow \max(score, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda})$
  end
end
Output: Split and default directions with max gain

1. 对于每个特征，根据特征值对每个值非 0 样本进行升序排序
2. 按照 `pre-sorted` 或 `histogram-based` 方法进行遍历，分别计算把 0 值样本全划分至左叶和右叶的信息增益
3. 选择最大信息增益对应的划分点

> 复杂度减小，相应的 $O(\#data) \rightarrow O(\#not\_none\_data)$

性能展示

分类/排序性能 & 训练速度

Table 3: Comparison of Exact Greedy Methods with 500 trees on Higgs-1M data.

Method	Time per Tree (sec)	Test AUC
XGBoost	0.6841	0.8304
XGBoost (colsample=0.5)	0.6401	0.8245
scikit-learn	28.51	0.8302
R.gbm	1.032	0.6224

Table 4: Comparison of Learning to Rank with 500 trees on Yahoo! LTRC Dataset

Method	Time per Tree (sec)	NDCG@10
XGBoost	0.826	0.7892
XGBoost (colsample=0.5)	0.506	0.7913
pGBRT [22]	2.576	0.7915

LightGBM: A Highly Efficient Gradient Boosting Decision Tree, 2017

LightGBM (Light Gradient Boosting Machine) 是柯国霖所在微软团队开发的一款开源机器学习项目，在 XGBoost 的基础上进一步做了许多改进：

- 每次划分时不需要遍历每个样本、每个特征值，在不几乎不损伤准确率的情况下进一步加速训练

算法层面

候选点信息增益指标

$$V_{j|O}(d) = \frac{1}{n_O}\left(\frac{(\sum_{\{x_i\in O:x_{ij}\le d\}} g_i)^2}{n^j_{l|O}(d)} + \frac{(\sum_{\{x_i\in O:x_{ij}> d\}} g_i)^2}{n^j_{r|O}(d)}\right)$$

> 信息增益指标为梯度的方差

连续特征值离散化

Algorithm 1: Histogram-based Algorithm

Input: I: training data, d: max depth
Input: m: feature dimension
$nodeSet \leftarrow \{0\}$ ▷ tree nodes in current level
$rowSet \leftarrow \{\{0, 1, 2, ...\}\}$ ▷ data indices in tree nodes
for $i = 1$ ***to*** d **do**
  for $node$ ***in*** $nodeSet$ **do**
    usedRows $\leftarrow rowSet[node]$
    for $k = 1$ ***to*** m **do**
      $H \leftarrow$ new Histogram()
      ▷ Build histogram
      for j ***in*** $usedRows$ **do**
        bin $\leftarrow I$.f[k][j].bin
        H[bin].y $\leftarrow$ H[bin].y + I.y[j]
        H[bin].n $\leftarrow$ H[bin].n + 1
      Find the best split on histogram H.
      ...
  Update $rowSet$ and $nodeSet$ according to the best split points.
  ...

1. 与 XGBoost 的 histogram-based 算法不同的是，数据的值已经离散化为 bin 的边界，训练过程中不再连续化，因此不需要索引样本数据
2. 只在最初的时候记录 bin 的梯度值之和 g，样本个数 n 用于信息增益计算
3. 划分后不用重新遍历，`H(right)=H(parent)-H(right)`

> 复杂度减小，相应的 $O(\#data * \#feature) \rightarrow O(\#bin * \#feature)$

GOSS：Gradient-based One-Side Sampling

基于信息增益指标，直觉上来讲绝对值大的梯度对该值的影响较大，因此提出了 GOSS 减少#data

Algorithm 2: Gradient-based One-Side Sampling

Input: I: training data, d: iterations
Input: a: sampling ratio of large gradient data
Input: b: sampling ratio of small gradient data
Input: $loss$: loss function, L: weak learner

models ← { }, fact ← $\frac{1-a}{b}$
topN ← a × len(I) , randN ← b × len(I)
for $i = 1$ ***to*** d **do**
  preds ← models.predict(I)
  g ← $loss(I$, preds), w ← {1,1,...}
  sorted ← GetSortedIndices(abs(g))
  topSet ← sorted[1:topN]
  randSet ← RandomPick(sorted[topN:len(I)], randN)
  usedSet ← topSet + randSet
  w[randSet] × = fact ▷ Assign weight $fact$ to the small gradient data.
  newModel ← L(I[usedSet], − g[usedSet], w[usedSet])
  models.append(newModel)

1. 基于上一轮模型计算所有样本的梯度值
2. 保留大梯度值样本 `topSet`，占总比 a%
3. 随机采样小梯度值样本，采样量占总训练样本 b%

> 此时信息增益计算方法为

$$\tilde{V}_j(d) = \frac{1}{n}\left(\frac{(\sum_{x_i \in A_l} g_i + \frac{1-a}{b}\sum_{x_i \in B_l} g_i)^2}{n_l^j(d)} + \frac{(\sum_{x_i \in A_r} g_i + \frac{1-a}{b}\sum_{x_i \in B_r} g_i)^2}{n_r^j(d)}\right)$$

4. 基于 `topSet + randSet` 训练新一轮迭代模型

> 复杂度减小，相应的 $O(\#data) \rightarrow O(\#top\_data + \#rand\_data)$

EFB：Exclusive Feature Bundling

高维特征非常稀疏，捆绑多个互相独立的特征为一个特征能加快模型训练，提出 EFB 减小#feature

Algorithm 3: Greedy Bundling

Input: F: features, K: max conflict count
Construct graph G
searchOrder ← G.sortByDegree()
bundles ← { }, bundlesConflict ← { }
for i ***in*** *searchOrder* **do**
  needNew ← True
  for $j = 1$ ***to*** *len(bundles)* **do**
    cnt ← ConflictCnt(bundles[j],F[i])
    if *cnt + bundlesConflict[i]* $\leq K$ **then**
      bundles[j].add(F[i]), needNew ← False
      break
  if *needNew* **then**
    Add $F[i]$ as a new bundle to *bundles*
Output: *bundles*

Algorithm 4: Merge Exclusive Features

Input: $numData$: number of data
Input: F: One bundle of exclusive features
binRanges ← {0}, totalBin ← 0
for f ***in*** F **do**
  totalBin += f.numBin
  binRanges.append(totalBin)
newBin ← new Bin(numData)
for $i = 1$ ***to*** $numData$ **do**
  newBin[i] ← 0
  for $j = 1$ ***to*** *len*(F) **do**
    if $F[j].bin[i] \neq 0$ **then**
      newBin[i] ← F[j].bin[i] + binRanges[j]
Output: $newBin$, $binRanges$

Algo 3: 收集(误差容许范围内)互相独立的特征

1. 构建无向图，顶点为特征(K 个)，边的权值为遍历样本后特征间不互相独立 (同时为 0)的次数
2. 根据顶点的度逆序排序
3. 遍历已有的 `bundles`，

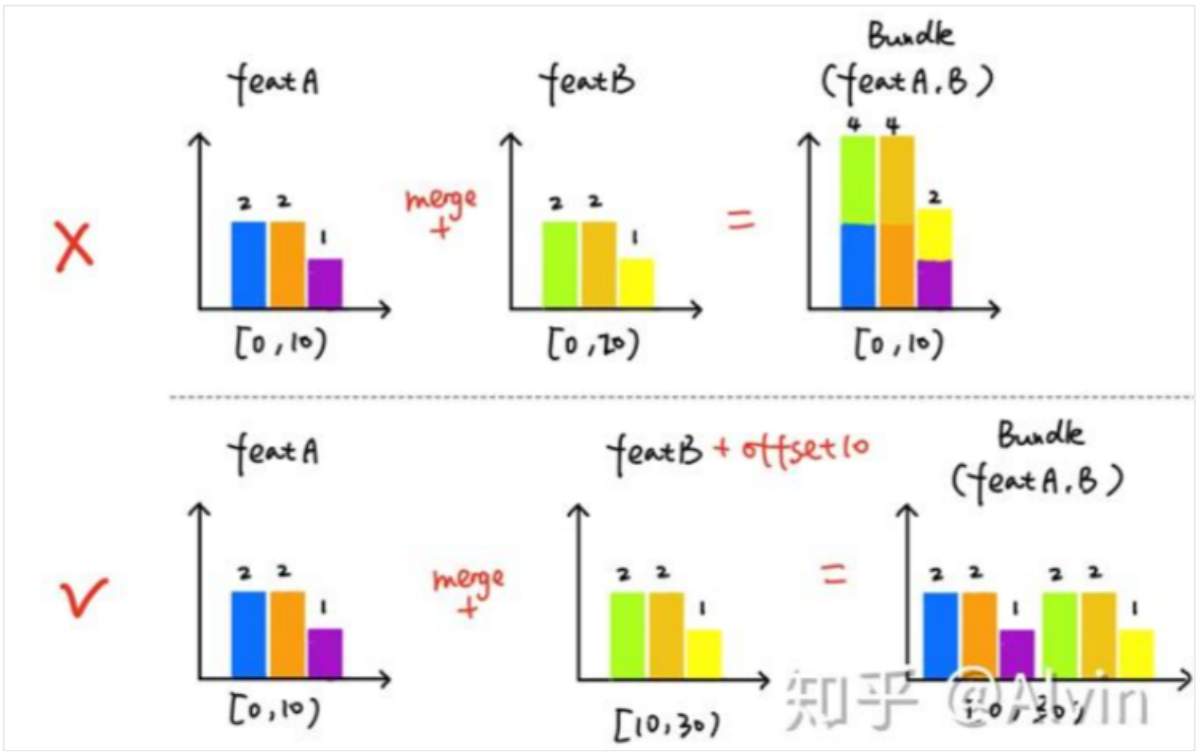
a. 在误差容许范围内，加入相应 bundle，

b. 不在误差容许范围内，新建 bundle 并放入

4. 返回捆绑特征集合

Algo 4: 合并互相独立特征中的集合为单个特征

1. 若一个特征需要被合并，则根据取值范围增加合并特征偏移量，得到被合并后的特征值
2. 最终分布在不同的 bin 中



复杂度减小，相应的 $O(\#data * \#feature) \rightarrow O(\#data * \#bundle)$

性能展示

训练速度

Table 2: Overall training time cost comparison. LightGBM is lgb_baseline with GOSS and EFB. EFB_only is lgb_baseline with EFB. The values in the table are the average time cost (seconds) for training one iteration.

a+b=1且使用了histogram bins排序

	xgb_exa	xgb_his	lgb_baseline	EFB_only	**LightGBM**
Allstate	10.85	2.63	6.07	0.71	**0.28**
Flight Delay	5.94	1.05	1.39	0.27	**0.22**
LETOR	5.55	0.63	0.49	0.46	**0.31**
KDD10	108.27	OOM	39.85	6.33	**2.85**
KDD12	191.99	OOM	168.26	20.23	**12.67**

映射连续值至离散值较遍历训练集、GOSS、EFB 能加快训练时间并加速

分类/排序性能

Table 3: Overall accuracy comparison on test datasets. Use AUC for classification task and NDCG@10 for ranking task. SGB is lgb_baseline with Stochastic Gradient Boosting, and its sampling ratio is the same as LightGBM.

	xgb_exa	xgb_his	lgb_baseline	SGB	**LightGBM**
Allstate	0.6070	0.6089	0.6093	0.6064 ± 7e-4	**0.6093 ± 9e-5**
Flight Delay	0.7601	0.7840	0.7847	0.7780 ± 8e-4	**0.7846 ± 4e-5**
LETOR	0.4977	0.4982	0.5277	0.5239 ± 6e-4	**0.5275 ± 5e-4**
KDD10	0.7796	OOM	0.78735	0.7759 ± 3e-4	**0.78732 ± 1e-4**
KDD12	0.7029	OOM	0.7049	0.6989 ± 8e-4	**0.7051 ± 5e-5**

LightGBM 性能较 XGBoost 要表现得稍好

EFB、GOSS 对模型性能几乎没有损失，甚至有提升 (表现为缓解了过拟合)

常见的缓解过拟合操作

- column subsampling: 训练特征部分采样，类似于 dropout
- shrinkage：分类器权重衰减，类似于 lr，即 $C(x)=\sum_{m=1}^{M} w_m C_m(x)$ 中随着迭代轮次增加，模型权重越小
- regularization：正则化项，如 $\gamma T+\frac{1}{2}\|w\|^2$
- GOSS
- `min_data_in_leaf`
- `max_depth`

MISC

CHI

CHI (CHI-square) 即卡方，用于筛选出和类别相关性强的特征词，去除或降低噪声特征

$$\chi^2(t_k,c_i)=\frac{N_d*(A*D-C*B)^2}{(A+C)*(B+D)}*\frac{1}{(A+B)*(C+D)}$$

t_k：特征词

c_i：目标类别

N_d：总样本数

A：属于c_i类 and 含特征词t_k的样本数

B：不属于c_i类 and 含特征词t_k的样本数

C：属于c_i类 and 不含有特征词t_k的样本数

D：不属于c_i类 and 不含特征词t_k的样本数

- 单个特征词的卡方值数目=类别数，选取chi值的方式有以下方法
  - `max`，更具有代表性，更能表达出一个特征对分类的决定性
  - `avg`
  - `tgt`，选取正例的chi值作为最终chi值
- $A*D-C*B$ 的值表示特征词与类别的相关性
  - `>0` 值越大，特征词对类别判定越重要
  - `<0` 值越小，特征词对类别判定越不重要

    > 由于进行了平方，平方后的值反而为大正数，可遏制此现象，比如置为0

> 能够一定程度上增加模型的精度，可作为一个调参对象

xgb 是宽度树，即 level-wise，而 lgb 是深度树，即 leaf-wise

- 【务实基础】LightGBM - 知乎 (zhihu.com)
- 机器学习 | 无痛看懂 LightGBM 源论文(附中英 PDF 版本下载) - 知乎 (zhihu.com)