LightGBM原理之论文详解

立刻有 机器学习算法那些事 2019-02-17

> 作者：立刻有
> 链接：
> https://blog.csdn.net/shine19930820/article/details/79123216
> 编辑：石头
> 原文：LightGBM : A Highly Efficient Gradient Boosting Decision Tree
> 原文下载：后台回复“LightGBM"

【Abstract】

Gradient Boosting Decision Tree (GBDT)非常流行却鲜有实现，只有像XGBoost和pGBRT。当特征维度较高和数据量巨大的时候，其实现中仍然存在效率和可扩展性的问题。一个主要原因就是对于每一个特征的每一个分裂点，都需要遍历全部数据计算信息增益，这一过程非常耗时。针对这一问题，本文提出两种新方法：Gradient-based One-Side Sampling (GOSS) 和Exclusive Feature Bundling (EFB)（基于梯度的one-side采样和互斥的特征捆绑）。在GOSS中，我们排除了重要的比例-具有小梯度的实例，只用剩下的来估计信息增益，我们证明，这些梯度大的实例在计算信息增益中扮演重要角色，GOSS可以用更小的数据量对信息增益进行相当准确的估计。对于EFB，我们捆绑互斥的特征（什么是互斥特征：例如特征间很少同时非零。），来降低特征的个数。我们证明完美地捆绑互斥特征是NP难的，但贪心算法能够实现相当好的逼近率，因此我们能够在不损害分割点准确率许多，有效减少特征的数量。（牺牲一点分割准确率降低特征数量），这一算法命名为LightGBM。在多个公共数据集实验证明，LightGBM加速了传统GBDT训练过程20倍以上，同时达到了几乎相同的精度 。

1. Introduction

GBDT因为其的有效性、准确性、可解释性，成为了广泛使用的机器学习算法。GBDT在许多机器学习任务上取得了最好的效果（ state-of-the-art），例如多分类，点击预测，排序。但最近几年随着大数据的爆发（特征量和数据量），GBDT面临平衡准确率和效率的调整。

GBDT缺点：对于每一个特征的每一个分裂点，都需要遍历全部数据来计算信息增益。因此，其计算复杂度将受到特征数量和数据量双重影响，造成处理大数据时十分耗时。

解决这个问题的直接方法就是减少特征量和数据量而且不影响精确度，有部分工作根据数据权重采样来加速boosting的过程，但由于gbdt没有样本权重不能应用。而本文提出两种新方法实现此目标。

Gradient-based One-Side Sampling (GOSS)：GBDT虽然没有数据权重，但每个数据实例有不同的梯度，根据计算信息增益的定义，梯度大的实例对信息增益有更大的影响，因此在下采样时，我们应该尽量保

留梯度大的样本（预先设定阈值，或者最高百分位间），随机去掉梯度小的样本。我们证明此措施在相同的采样率下比随机采样获得更准确的结果，尤其是在信息增益范围较大时。

Exclusive Feature Bundling (EFB)：通常在真实应用中，虽然特征量比较多，但是由于特征空间十分稀疏，是否可以设计一种无损的方法来减少有效特征呢？特别在稀疏特征空间上，许多特征几乎是互斥的（例如许多特征不会同时为非零值，像one-hot），我们可以捆绑互斥的特征。最后，我们将捆绑问题归约到图着色问题，通过贪心算法求得近似解。

2. Preliminaries

2.1 GBDT and Its Complexity Analysis

GBDT是一种集成模型的决策树，顺序训练决策树。每次迭代中，GBDT通过拟合负梯度（残差）来学到决策树。

学习决策树是GBDT主要的时间花销，而学习决策树中找到最优切分点最消耗时间。广泛采用的预排序算法来找到最优切分点，这种方法会列举预排序中所有可能的切分点。这种算法虽然能够找到最优的切分点，但对于训练速度和内存消耗上都效率低。另一种流行算法是直方图算法（histogram-based algorithm）。直方图算法并不通过特征排序找到最优的切分点，而是将连续的特征值抽象成离散的分箱，并使用这些分箱在训练过程中构建特征直方图，这种算法更加训练速度和内存消耗上都更加高效，lightGBM使用此种算法。

histogram-based算法通过直方图寻找最优切分点，其建直方图消耗O(#data * #feature)，寻找最优切分点消耗O(#bin * # feature)，而#bin的数量远小于#data，所以建直方图为主要时间消耗。如果能够减少数据量或特征量，那么还能够够加速GBDT的训练。

2.2 Related Work

GBDT有许多实现，如XGBoost，PGBRT，Scikit-learn，gbm in R。Scikit-learn和gbm in R实现都用了预排序，pGBRT使用了直方图算法，XGBoost支持预排序和直方图算法，由于XGBoost胜过其他算法，我们用它作为baseline。

为了减小训练数据集，通常做法是下采样。例如过滤掉权重小于阈值的数据。SGB每次迭代中用随机子集训练弱学习器。或者采样率基于训练过程动态调整。除了基于AdaBoost的SGB不能直接应用于GBDT，因为GBDT中没有原始的权重。虽然SGB也能间接应用于GBDT，但往往会影响精度。

同样，过滤掉弱特征（什么是弱特征）来减少特征量。通常用主成分分析或者投影法。当然，这些方法依赖于一个假设-特征包含高度的冗余，但实际中往往不是。（设计特征来自于其独特的贡献，移除任一维度都可以某种程度上影响精度）。

实际中大规模的数据集通常都是非常稀疏的，使用预排序算法的GBDT能够通过无视为0的特征来降低训练时间消耗。然后直方图算法没有优化稀疏的方案。因为直方图算法无论特征值是否为0，都需要为每个数据检索特征区间值。如果基于直方图的GBDT能够有效利用稀疏特征将是最优。

下图是两个算法的对比：

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.
  ...

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

3. Gradient-based One-Side Sampling

GOSS是一种在减少数据量和保证精度上平衡的算法。

3.1 Algorithm Description

AdaBoost中，样本权重是数据实例重要性的指标。然而在GBDT中没有原始样本权重，不能应用权重采样。幸运的事，我们观察到GBDT中每个数据都有不同的梯度值，对采样十分有用，即实例的梯度小，实例训练误差也就较小，已经被学习得很好了，直接想法就是丢掉这部分梯度小的数据。然而这样做会改变数据的分布，将会影响训练的模型的精确度，为了避免此问题，我们提出了GOSS。

GOSS保留所有的梯度较大的实例，在梯度小的实例上使用随机采样。为了抵消对数据分布的影响，计算信息增益的时候，GOSS对小梯度的数据引入常量乘数。GOSS首先根据数据的梯度绝对值排序，选取top a个实例。然后在剩余的数据中随机采样b个实例。接着计算信息增益时为采样出的小梯度数据乘以(1-a)/b，这样算法就会更关注训练不足的实例，而不会过多改变原数据集的分布。

3.2 Theoretical Analysis

GBDT使用决策树，来学习获得一个将输入空间映射到梯度空间的函数。假设训练集有n个实例x1，...，xn，特征维度为s。每次梯度迭代时，模型数据变量的损失函数的负梯度方向为g1，...，gn，决策树通过最优切分点（最大信息增益点）将数据分到各个节点。GBDT通过分割后的方差衡量信息增益。

定义3.1：O表示某个固定节点的训练集，分割特征 j 的分割点d定义为：

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

其中，

$$n_O = \sum I[x_i \in O], n^j_{l|O}(d) = \sum I[x_i \in O : x_{ij} \le d] \text{ and } n^j_{r|O}(d) = \sum I[x_i \in O : x_{ij} > d].$$

遍历每个特征分裂点 j，找到

$$d^*_j = argmax_d V_j(d)$$

对应最大的分裂节点，并计算最大的信息增益$V_j(d^*_j)$。然后，将数据根据特征j^*的分裂点d_{j^*}将数据分到左右节点。

在GOSS中，

1. 首先根据数据的梯度将训练降序排序。
2. 保留top a个数据实例，作为数据子集A。
3. 对于剩下的数据的实例，随机采样获得大小为b的数据子集B。
4. 最后我们通过以下方程估计信息增益:

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

此处GOSS通过较小的数据集估计信息增益$\tilde{V}_j(d)$，将大大地减小计算量。更重要的是，我们接下来理论表明GOSS不会丢失许多训练精度，胜过（outperform）随机采样。理论的证明再附加材料。

Theorem 3.2 我们定义GOSS近似误差为：$\mathcal{E}(d) = |\tilde{V}_j(d) - V_j(d)|$，且

$$\bar{g}^i_l(d) = \frac{\sum_{x_i \in (A \cup A^c)_l} |g_i|}{n^j_l(d)},$$

$$\bar{g}^i_r(d) = \frac{\sum_{x_i \in (A \cup A^c)_r} |g_i|}{n^j_r(d)}$$

概率至少是1−δ，有：

$$\mathcal{E}(d) \le C^2_{a,b} \ln 1/\delta \cdot \max\left\{\frac{1}{n^j_l(d)}, \frac{1}{n^j_r(d)}\right\} + 2DC_{a,b}\sqrt{\frac{\ln 1/\delta}{n}},$$

其中：

$$C_{a,b} = \frac{1-a}{\sqrt{b}} \max_{x_i \in A^c} |g_i|,$$

$$D = \max(\bar{g}_l^j(d), \bar{g}_r^j(d))$$

根据理论3.2，我们得出以下结论：

1. GOSS的渐进逼近比率是：

$$\mathcal{O}\left(\frac{1}{n_l^j(d)} + \frac{1}{n_r^j(d)} + \frac{1}{\sqrt{n}}\right)$$

如果数据分割不是极不平衡，比如：

$$n_l^j(d) \geq \mathcal{O}(\sqrt{n}) \text{ 且 } n_r^j(d) \geq \mathcal{O}(\sqrt{n})$$

近似误差主要由第二项主导，当*n*趋于无穷（数据量很大）时，

$\mathcal{O}(\sqrt{n})$趋于0。即数据量越大，误差越小，精度越高。

2. 随机采样是GOSS在a=0的一种情况。多数情况下，GOSS性能优于随机采样，即以下情况：

$$C_{0,\beta} > C_{a,\beta-a}$$

即：

$$\frac{\alpha_a}{\sqrt{\beta}} > \frac{1-a}{\sqrt{\beta-a}}$$

其中，

$$\alpha_a = \max_{x_i \in A \cup A^c} |g_i| / \max_{x_i \in A^c} |g_i|$$

下面分析GOSS的**泛化性**。考虑GOSS泛化误差：

$$\mathcal{E}_{gen}^{GOSS}(d) = |\tilde{V}_j(d) - V_*(d)|$$

这是GOSS抽样的的实例计算出的方差增益与实际样本方差增益之间的差距。

变换为：

$$\mathcal{E}_{gen}^{GOSS}(d) \leq |\tilde{V}_j(d) - V_j(d)| + |V_j(d) - V_*(d)| \triangleq \mathcal{E}_{GOSS}(d) + \mathcal{E}_{gen}(d)$$

因此，在GOSS准确的情况下，GOSS泛化误差近似于全数据量的真实数据。另一方面，采样将增加基学习器的多样性（因为每次采样获得的数据可能会不同），这将提高泛化性。

4. Exclusive Feature Bundling

这一章介绍如何有效减少特征的数量

高维的数据通常是稀疏的，这种稀疏性启发我们设计一种无损地方法来减少特征的维度。特别的，稀疏特征空间中，许多特征是互斥的，例如他们从不同时为非零值。我们可以绑定互斥的特征为单一特征，通过仔细设计特征搜寻算法，我们从特征捆绑中构建了与单个特征相同的特征直方图。这种方式的间直方图时间复杂度从O(#data * #feature)降到O(#data * #bundle)，由于#bundle << # feature，我们能够极大地加速GBDT的训练过程而且损失精度。

有两个问题：

1. 怎么判定那些特征应该绑在一起（build bundled）？

2. 怎么把特征绑为一个（merge feature）？

4.1 bundle（什么样的特征被绑定）？

理论 4.1：将特征分割为较小量的互斥特征群是NP难的。

证明：将图着色问题归约为此问题，而图着色是NP难的，所以此问题就是NP难的。

> 给定图着色实例G=(V, E)。以G的关联矩阵的每一行为特征，得到我们问题的一个实例有|V|个特征。 很容易看到，在我们的问题中，一个独特的特征包与一组具有相同颜色的顶点相对应，反之亦然。

理论4.1说明多项式时间中求解这个NP难问题不可行。为了寻找好的近似算法，我们将最优捆绑问题归结为图着色问题，如果两个特征之间不是相互排斥，那么我们用一个边将他们连接，然后用合理的贪婪算法（具有恒定的近似比）用于图着色来做特征捆绑。 此外，我们注意到通常有很多特征，尽管不是100％相互排斥的，也很少同时取非零值。 如果我们的算法可以允许一小部分的冲突，我们可以得到更少的特征包，进一步提高计算效率。经过简单的计算，随机污染小部分特征值将影响精度最多为：

$$\mathcal{O}([(1-\gamma)n]^{-2/3})$$

γ是每个绑定中的最大冲突比率，当其相对较小时，能够完成精度和效率之间的平衡。

算法3：基于上面的讨论，我们设计了算法3，伪代码见下图，具体算法：

1.建立一个图，每个点代表特征，每个边有权重，其权重和特征之间总体冲突相关。

2. 按照降序排列图中的度数来排序特征。

3. 检查排序之后的每个特征，对他进行特征绑定或者建立新的绑定使得操作之后的总体冲突最小。

算法3的时间复杂度是O(#feature^2)，训练之前只处理一次，其时间复杂度在特征不是特别多的情况下是可以接受的，但难以应对百万维的特征。为了继续提高效率，我们提出了一个更加高效的无图的排序策略：将特征按照非零值个数排序，这和使用图节点的度排序相似，因为更多的非零值通常会导致冲突，新算法在算法3基础上改变了排序策略。

Algorithm 3: Greedy Bundling

Input: F: features, K: max conflict count
Construct graph G
searchOrder $\leftarrow$ G.sortByDegree()
bundles $\leftarrow$ {}, bundlesConflict $\leftarrow$ {}
for i ***in*** *searchOrder* **do**
  needNew $\leftarrow$ True
  for $j = 1$ ***to*** *len(bundles)* **do**
    cnt $\leftarrow$ ConflictCnt(bundles[j],F[i])
    if *cnt + bundlesConflict[i]* $\leq K$ **then**
      bundles[j].add(F[i]), needNew $\leftarrow$ 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 $\leftarrow$ {0}, totalBin $\leftarrow$ 0
for f ***in*** F **do**
  totalBin += f.numBin
  binRanges.append(totalBin)
newBin $\leftarrow$ new Bin(numData)
for $i = 1$ ***to*** $numData$ **do**
  newBin[i] $\leftarrow$ 0
  for $j = 1$ ***to*** *len*(F) **do**
    if $F[j].bin[i] \neq 0$ **then**
      newBin[i] $\leftarrow$ F[j].bin[i] + binRanges[j]
Output: $newBin, binRanges$

4.2 merging features(特征合并)

如何合并同一个bundle的特征来降低训练时间复杂度。关键在于原始特征值可以从bundle中区分出来。鉴于直方图算法存储离散值而不是连续特征值，我们通过将互斥特征放在不同的箱中来构建bundle。这可以通过将偏移量添加到特征原始值中实现，例如，假设bundle中有两个特征，原始特征A取值[0, 10]，B取值[0, 20]。我们添加偏移量10到B中，因此B取值[10, 30]。通过这种做法，就可以安全地将A、B特征合并，使用一个取值[0, 30]的特征取代AB。算法见算法4，

EFB算法能够将许多互斥的特征变为低维稠密的特征，就能够有效的避免不必要0值特征的计算。实际，通过用表记录数据中的非零值，来忽略零值特征，达到优化基础的直方图算法。通过扫描表中的数据，建直方图的时间复杂度将从O(#data)降到O(#non_zero_data)。当然，这种方法在构建树过程中需要而额外的内存和计算开销来维持预特征表。我们在lightGBM中将此优化作为基本函数，因为当bundles是稀疏的时候，这个优化与EFB不冲突（可以用于EFB）。

5. Experiments

这部分主要写了lightGBM的实验结果，主要用了五个公开数据集，如下

Table 1: Datasets used in the experiments.

Name	$\#data$	$\#feature$	Description	Task	Metric
Allstate	12 M	4228	Sparse	Binary classification	AUC
Flight Delay	10 M	700	Sparse	Binary classification	AUC
LETOR	2M	136	Dense	Ranking	NDCG [4]
KDD10	19M	29M	Sparse	Binary classification	AUC
KDD12	119M	54M	Sparse	Binary classification	AUC

微软的排序数据集（LETOR）包括30K的网页搜索，数据集几乎都是稠密的数值特征。
Allstate是保险和航空延误数据都包含了大量的one-hot特征。
后两个是KDD CUP2010和KDD CPU2012数据集，使用了冠军解决方案中的特征，其中包含了稀疏和稠密的特征，并且这两个数据集特别大。
这些数据集都比较大，而且包含了稀疏和稠密的特征，涵盖了很多真实的业务，因此它们能够完全地测试lightGBM的性能。

5.1 Overall Comparison

XGBoost和lightGBM without GOSS 和EFB（lgb baseline），作为比较的基准。XGBoost使用了两个版本：xgb_exa(预排序)和xgb_his(直方图算法)。对于xgb_exa做了参数调整，使XGBoost长成和其他算法相似的树。并且调整参数在速度和准确率间平衡。对于Allstate、KDD10 和 KDD2012，设置a = 0.05，b = 0.05 ，对于航空延误和LETOR，我们设置a = 0.1，b = 0.1 ，数据集EFB我们设置γ = 0 。所有的算法有固定迭代次数。在一定迭代次数内，我们取最高的分数。

下表是训练时间对比（时间是每次迭代的平均时间）

	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**

下表是精确度对比，分类使用AUC评价，排序使用NDCG评价。

	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**

下图是飞行延时和LETOR的训练曲线。

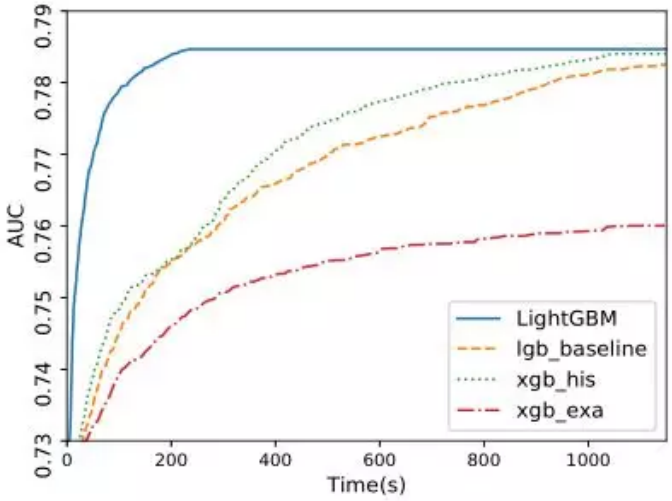

Figure 1: Time-AUC curve on Flight Delay.

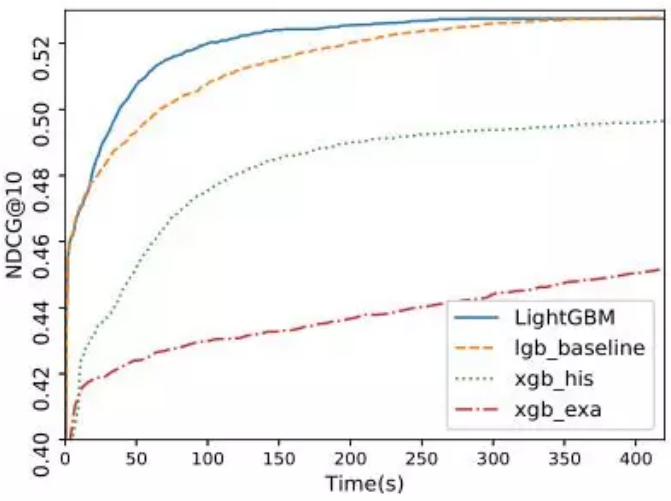

Figure 2: Time-NDCG curve on LETOR.

5.2 Analysis on GOSS

速度上：GOSS具有加速训练的能力，表2中可以看出GOSS几乎加速了两倍。虽然GOSS用了10%-20%的数据训练，但是由于有一些额外的计算，所以加速并不和数据量成反比，但是GOSS仍然极大

加速了训练。

精度上：从表4中可以看出，同样采样率，GOSS精度总比SGB好。

5.3 Analysis on EFB

表2表明，EFB在大数据集上能够极大加速训练。因为EFB合并大量的稀疏特征到低维稠密的特征，并且由于之前的孤立的特征被bundle到一起，能能够极大提高缓存的命中率，因此，它全部的效率提高是动态的。

综上这些分析，EFB是改进直方图算法稀疏性的非常高效的算法，能够极大地加速GBDT训练。

6. Conclusion

本文提出了新颖的GBDT算法–LightGBM，它包含了连个新颖的技术：Gradient-based One-Side Sampling (GOSS) 和Exclusive Feature Bundling (EFB)（基于梯度的one-side采样和互斥的特征捆绑）来处理大数据量和高维特征的场景。我们在理论分析和实验研究表明，GOSS和EFB使得LightGBM在计算速度和内存消耗上明显优于XGBoost和SGB。
未来，我们将研究优化如何在GOSS中选择a，b。继续提高EFB在高维特征上的性能，无论其是否是稀疏的。

推荐阅读

LightGBM算法原理小结