深入理解GBDT、XGBoost、LightGBM系列(三)

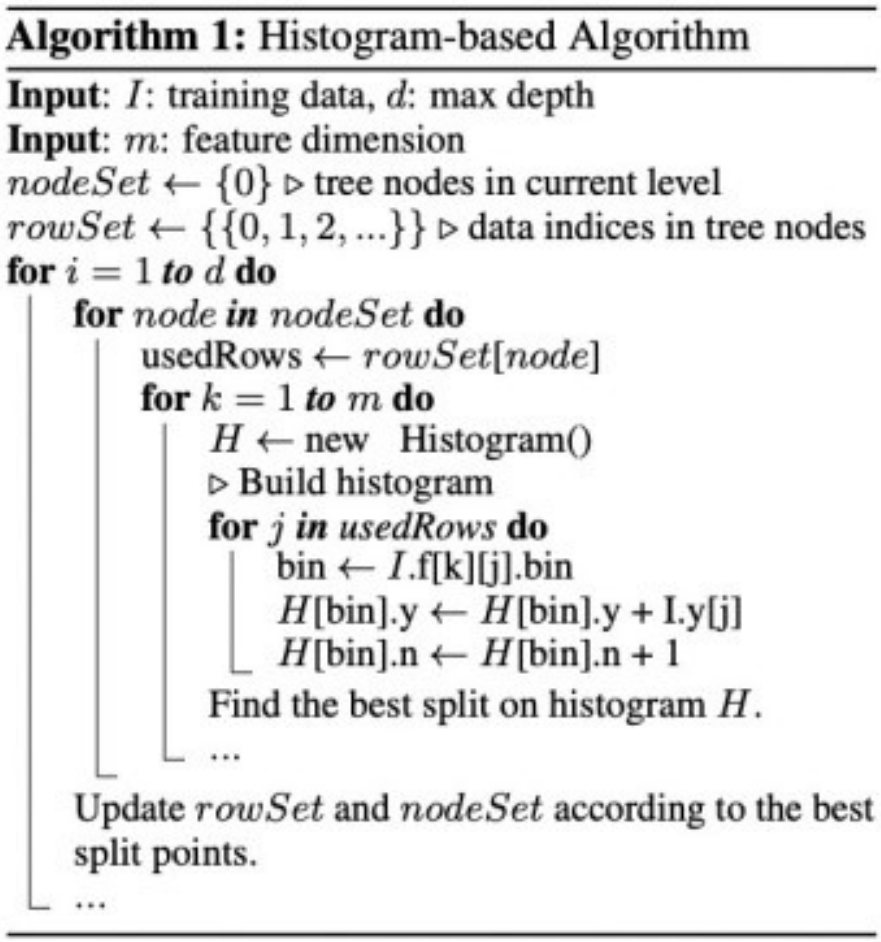
有了前两篇的基础，再来理解LightGBM就比较简单了，可以说LightGBM就是在XGBoost基础上的再优化，主要解决在数据量大、特征维度高的情况下效率和扩展性不足的问题。造成这个问题原因是对每一个特征在决策每一个分裂点时需要遍历所有的数据来估计树结构的得分，这是非常耗时的。

为了解决这个问题，在论文中提出了两个技术点，一是Gradient-based One-side Sampling (GOSS)，用于解决数据量大的问题，二是 Exclusive Feature Bundling (EFB)，用于解决特征维度高的问题。下面分别来看看这两个技术。

GOSS

提出GOSS是由于作者注意到不同梯度的样本在计算树结构的分值时起到的作用是不同的。梯度大的数据样本对分值的贡献更大，梯度小的样本训练误差也小，这些样本已经训练的很好了。直观的想法就是直接丢弃这些梯度小的样本，但这会改变数据的分布，降低模型的训练精度。所以，GOSS的下采样方式是保留梯度大的样本，对梯度小的样本进行随机采样。为了补偿采样对数据分布的影响，在计算树的分值时，给这些梯度小的样本乘了一个常数。

具体的做法是，首先对样本依据梯度带下进行降序排序，选择前a%的数据，然后对剩余的数据采样b%，进而对采样数据进行增强的的常数为 $\frac{1-a}{b}$ 。这样模型就更加关注训练不足的样本，但对原始的数据分布影响也不大。具体的算法流程如下图。



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)

作者在论文中证明了这样的处理方式相对于完全随机采样能够获得更好的分值估计，尤其是分值的波动范围较大时。证明的公式实在有点复杂，没耐力都看完，想看的朋友自己去看论文吧(参考资料[1])。

读到这里脑子里闪过的一个问题：“这里区分梯度大小的阈值证明定？”，也就是a选择多少，b选择多少合适。

EFB

EFB(Exclusive Feature Bundling)用于降低特征的维度，高维特征往往是稀疏的。在稀疏的特征空间中，许多特征是互斥，比如它们不会同时为非零值，这就允许把互斥的特征捆绑成一个特征。通过这种方式构建直方图的时间复杂度由 $O(\#data \times \#feature)$ 变为 $O(\#data \times \#bundle)$，如果bundle的量远小于feature的量，就能加速训练的过程。

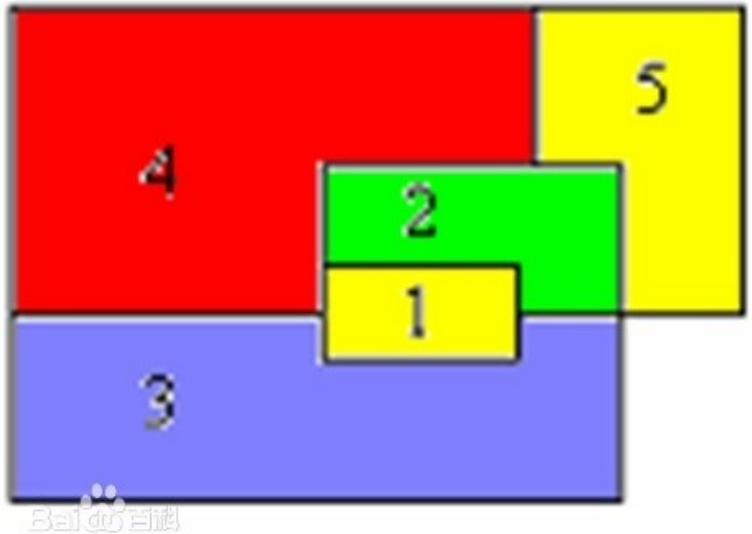
为了实现EFB，有两个问题需要解决，一是哪些特征应该进行捆绑，二是如何进行捆绑。

对于哪些特征应该捆绑的问题，转化为图着色问题，特征作为顶点，给不互斥两个特征间加上边，然后通过贪婪算法即可获得需要的绑定(bundles)。

鉴于个人背景知识不足，简单介绍一下图着色问题，摘自参考资料[4]。

图着色问题（Graph Coloring Problem, GCP） 又称着色问题，是最著名的NP-完全问题之一。

数学定义：给定一个无向图G=（V, E），其中V为顶点集合，E为边集合，图着色问题即为将V分为K个颜色组，每个组形成一个独立集，即其中没有相邻的顶点。其优化版本是希望获得最小的K值。



图着色问题包含两个问题：

（1）图的m-着色判定问题——给定无向连通图G和m种不同的颜色。用这些颜色为图G的各顶点着色，每个顶点着一种颜色，是否有一种着色法使G中任意相邻的2个顶点着不同颜色?

（2）图的m-着色优化问题——若一个图最少需要m种颜色才能使图中任意相邻的2个顶点着不同颜色，则称这个数m为该图的色数。求一个图的最小色数m的问题称为m-着色优化问题。

在LightGBM中用到的是问题(2)，有了图的色数m，也就有了特征的绑定方案，具体求解方法可以看参考资料[3]。

更进一步，有些特征之间不是百分之百互斥，但也很少同时出现非零值，如果允许有少量的冲突，最终获得的绑定数量会更少，计算效率更高。lightGBM中的绑定方法如下图中的算法3。

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: $newBin, binRanges$

在算法3中可见，LightGBM首先构建了一个边带有权值的图，这个权值表示特征之间的冲突程度。然后把特征按照权值降序排列，遍历序列中的每一个特征，决定是把它放入一个已有的冲突很小的绑定，还是创建一个新的绑定，这里的冲突程度大小由阈值 γ 决定。此时的时间复杂度是 $O(\#feature^2)$ 。

在实际的实现中并没有构建这个图，而是直接按特征中非零值的多少排序，这和按照冲突程度排序类似，因为非零值多的特征冲突的概率更大。

解决了哪些特征进行捆绑的问题，接下来解决如何捆绑的问题，这个问题的关键是必须确保原始特征能够从绑定中识别出来。基于直方图的算法处理的离散的分片而不是连续数值，所以LightGBM中捆绑的方式是把互斥的特征放到不同的分片中。比如特征A的范围是[0, 10)，特征B的范围是[0, 20)，给特征B加上10的偏移量，变为[10, 30)，合并后就成为一个[0, 30)的特征绑定，替代了原来的特征A、B。

EFB会把许多互斥的特征绑定为少量的稠密特征，这样就避免了大量的0特征值计算。在LightGBM的工程实践中，在基于直方图的算法中直接忽略了零特征值，用一个表记录了每个特征的非零特征值。这样做时间复杂度由 $O(\#data)$ 变为 $O(\#non_zero_data)$ ，但这样做需要额外的内存消耗。

至此LightGBM中的两个关键创新点就介绍完了，接下来看看实验结果。

实验结果

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.

	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**

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**

上面两个表是论文中的实验结果，xgb_exa是XGBoost pre-sorted算法，xgb_his是XGBoost histogram 算法，lgb_baseline表示不使用GOSS和EFB算法的LightGBM，表2是执行效率数据，表3是效果数据。由表2可见LightGBM在执行效率上确实有非常大的提升，由表3可见在模型效果上都有小幅度的提升。

在LightGBM中，GOSS不是默认选项，EFB是默认打开的，在实际使用时需要根据数据情况手动设置一下。

至于前面那个问题，保留大梯度数据比例a，和小梯度数据采样比例b，设置成多少是合理的，在论文的最后也有提到，这是一个future work。

至此，这个系列就完成了，从第一篇无人问津，到第二篇很多朋友的赞同，内心还是有点小窃喜，说明真的帮助到了一些朋友，希望这一篇也有帮助。

根据我的个人经验，第一篇才是理解原理的关键，理解了GBDT，再看后面的XGBoost、LightGBM，就会发现都是水到渠成。

希望这几篇文章可以帮朋友们理清思路，而不是去强背XGBoost的优缺点。

参考资料

[1]《LightGBM: A Highly Efficient Gradient Boosting Decision Tree》

[2]《不手写lightgbm（1）—怎么分桶的》

[3]

[4]