LightGBM: A Highly Efficient Gradient Boosting Decision Tree

GuolinKe,Qi Meng,Thomas Finley,
Taifeng Wang, Wei Chen,Weidong Ma,Qiwei Ye,Tie-Yan Liu
Microsoft Research
Peking University
Microsoft Redmond

Introduction

Gradient boosting decision tree (GBDT) is a widely-used machine learning algorithm, **due to its efficiency, accuracy, and interpretability.** GBDT achieves state-of-the-art performances in many machine learning tasks, such as multi-class classification, click prediction, and learning to rank. In recent years, with the emergence of big data (in terms of both the number of features and the number of instances), GBDT is facing new challenges, especially in the tradeoff between accuracy and efficiency. Conventional implementations of GBDT need to, for every feature, scan all the data instances to estimate the information gain of all the possible split points. Therefore, their computational complexities will be proportional to both the number of features and the number of instances. **This makes these implementations very time consuming when handling big data.**

In this paper, we propose two novel techniques:

Gradient-based One-Side Sampling (GOSS).

While there is no native weight for data instance in GBDT, we notice that data instances with different gradients play different roles in the computation of information gain. In particular, according to the definition of information gain, those instances with larger gradients will contribute more to the information gain. Therefore, when down sampling the data instances, in order to retain the accuracy of information gain estimation, **we should better keep those instances with large gradients, and only randomly drop those instances with small gradients.**

Exclusive Feature Bundling (EFB).

Usually in real applications, although there are a large number of features, the feature space is quite sparse, which provides us a possibility of designing a nearly lossless approach to reduce the number of effective features. Specifically, in a sparse feature space, many features are exclusive, i.e., they rarely take nonzero values simultaneously. **We can safely bundle such exclusive features.** To this end, we design an efficient algorithm by reducing the optimal bundling problem to a graph coloring problem, and solving it by a greedy algorithm with a constant approximation ratio.

Gradient-based One-Side Sampling (GOSS):

Algorithm 1: Histogram-based Algorithm

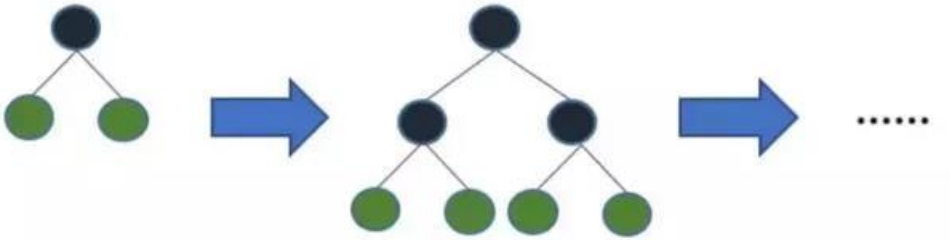
Input: I: training data, d: max depth
Input: m: feature dimension
$nodeSet \leftarrow \{0\}$ ▷ tree nodes in current level
$rowSet \leftarrow \{\{0, 1, 2, ...\}\}$ ▷ data indices in tree nodes
for $i = 1$ ***to*** d **do**
  for $node$ ***in*** $nodeSet$ **do**
    usedRows $\leftarrow rowSet[node]$
    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.
  ...

As shown in Alg. 1, the histogram-based algorithm finds the best split points based on the feature histograms.

It costs O(#data × #feature) for histogram building and O(#bin × #feature) for split point finding. Since #bin is usually much smaller than #data, histogram building will dominate the computational complexity.

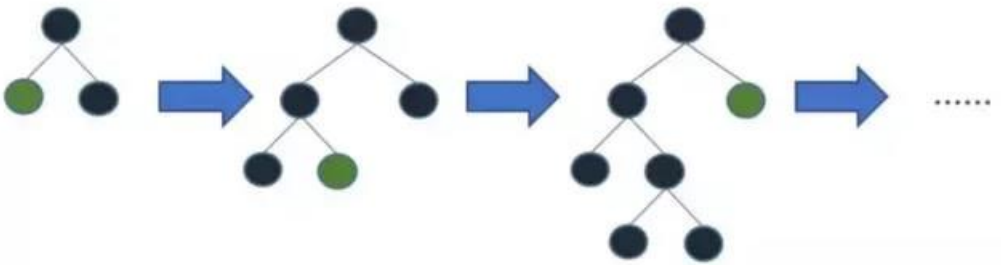
Algorithm 2: Gradient-based One-Side Sampling

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 $\triangleright$ Assign weight $fact$ to the small gradient data.
  newModel $\leftarrow$ L(I[usedSet], $-$ g[usedSet], w[usedSet])
  models.append(newModel)



Level-wise tree growth

LightGBM :



Leaf-wise tree growth

GBDT uses decision trees to learn a function from the input space $\mathcal{X}^s$ to the gradient space $\mathcal{G}$. Suppose that we have a training set with n i.i.d. instances $\{x_1, \cdots, x_n\}$, where each x_i is a vector with dimension s in space $\mathcal{X}^s$. In each iteration of gradient boosting, the negative gradients of the loss function with respect to the output of the model are denoted as $\{g_1, \cdots, g_n\}$. The decision tree model splits each node at the most informative feature (with the largest information gain). For GBDT, the information gain is usually measured by the variance after splitting, which is defined as below.

Definition 3.1 *Let O be the training dataset on a fixed node of the decision tree. The variance gain of splitting feature j at point d for this node is defined as*

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

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

For feature j, the decision tree algorithm selects $d^*_j = argmax_d V_j(d)$ and calculates the largest gain $V_j(d^*_j)$. Then, the data are split according feature j^* at point d_{j^*} into the left and right child nodes.

In our proposed GOSS method, first, we rank the training instances according to their absolute values of their gradients in the descending order; second, we keep the top-$a \times 100\%$ instances with the larger gradients and get an instance subset A; then, for the remaining set A^c consisting $(1-a) \times 100\%$ instances with smaller gradients, we further randomly sample a subset B with size $b \times |A^c|$; finally, we split the instances according to the estimated variance gain $\tilde{V}_j(d)$ over the subset $A \cup B$, i.e.,

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

where $A_l = \{x_i \in A : x_{ij} \leq d\}$,$A_r = \{x_i \in A : x_{ij} > d\}$,$B_l = \{x_i \in B : x_{ij} \leq d\}$,$B_r = \{x_i \in B : x_{ij} > d\}$, and the coefficient $\frac{1-a}{b}$ is used to normalize the sum of the gradients over B back to the size of A^c.

Theorem 3.2 *We denote the approximation error in GOSS as* $\mathcal{E}(d) = |\tilde{V}_j(d) - V_j(d)|$ *and* $\bar{g}_l^j(d) = \frac{\sum_{x_i \in (A \cup A^c)_l} |g_i|}{n_l^j(d)}$, $\bar{g}_r^j(d) = \frac{\sum_{x_i \in (A \cup A^c)_r} |g_i|}{n_r^j(d)}$. *With probability at least* $1-\delta$, *we have*

$$\mathcal{E}(d) \leq C_{a,b}^2 \ln 1/\delta \cdot \max\left\{\frac{1}{n_l^j(d)}, \frac{1}{n_r^j(d)}\right\} + 2DC_{a,b}\sqrt{\frac{\ln 1/\delta}{n}}, \tag{2}$$

where $C_{a,b} = \frac{1-a}{\sqrt{b}} \max_{x_i \in A^c} |g_i|$, *and* $D = \max(\bar{g}_l^j(d), \bar{g}_r^j(d))$.

We have the following discussions:

(1) The asymptotic approximation ratio of GOSS is $\mathcal{O}\left(\frac{1}{n_l^j(d)} + \frac{1}{n_r^j(d)} + \frac{1}{\sqrt{n}}\right)$.

If the split is not too unbalanced (i.e., $n_l^j(d) \geq \mathcal{O}(\sqrt{n})$ and $n_r^j(d) \geq \mathcal{O}(\sqrt{n})$), the approximation error will be dominated by the second term of Ineq.(2) which decreases to 0 in O(√n) with n → ∞. That means when number of data is large, the approximation is quite accurate.

(2) Random sampling is a special case of GOSS with a = 0. In many cases, GOSS could outperform random sampling, under the condition .

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

Exclusive Feature Bundling (EFB).

High-dimensional data are usually very sparse. The sparsity of the feature space provides us a possibility of designing a nearly lossless approach to reduce the number of features. Specifically, **in a sparse feature space, many features are mutually exclusive, i.e., they never take nonzero values simultaneously. We can safely bundle exclusive features into a single feature (which we call an exclusive feature bundle)**. By a carefully designed feature scanning algorithm, we can build the same feature histograms from the feature bundles as those from individual features. In this way, the complexity of histogram building changes from O(#data × #feature) to O(#data × #bundle),
while #bundle << #feature.

There are two issues to be addressed.
The first one is to determine which features should be bundled together.
(Greed Bundling)

The second is how to construct the bundle.
(Merge Exclusive Features)

Greed bundling

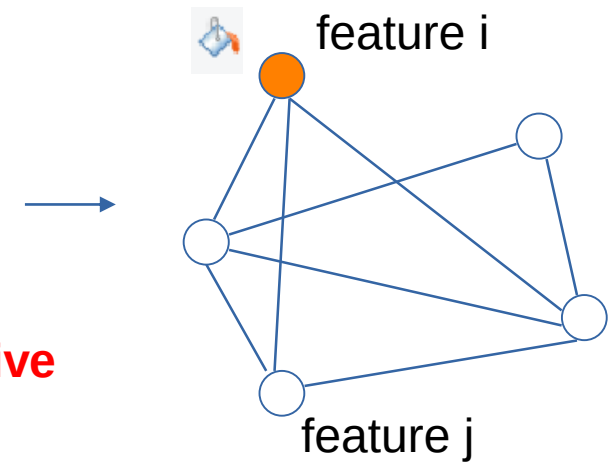
Theorem 4.1 *The problem of partitioning features into a smallest number of exclusive bundles is NP-hard.*

Proof: We will reduce the graph coloring problem [25] to our problem. Since graph coloring problem is NP-hard, we can then deduce our conclusion.

Given any instance $G = (V, E)$ of the graph coloring problem. We construct an instance of our problem as follows. Take each row of the incidence matrix of G as a feature, and get an instance of our problem with $|V|$ features. It is easy to see that an exclusive bundle of features in our problem corresponds to a set of vertices with the same color, and vice versa. □

feature1	feature2
0	0
1	0
1	1

NOT exclusive



Furthermore, we notice that **there are usually quite a few features, although not 100% mutually exclusive**, also rarely take nonzero values simultaneously. If our algorithm can allow a small fraction of conflicts, we can get an even smaller number of feature bundles and further improve the computational efficiency.

By simple calculation, random polluting a small fraction of feature values will affect the training accuracy by at most $\mathcal{O}([(1-\gamma)n]^{-2/3})$, where γ is the maximal conflict rate in each bundle.

So, if we choose a relatively small γ, we will be able to achieve a good balance between accuracy and efficiency.

Algorithm 3: Greedy Bundling

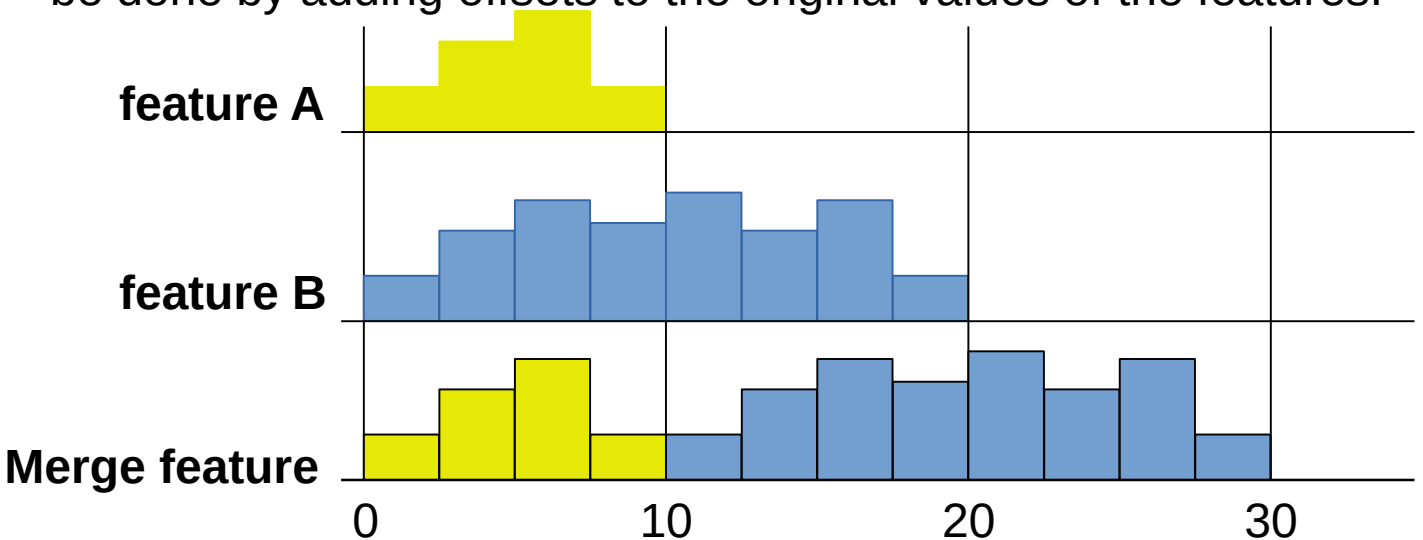
Input: F: features, K: max conflict count
Construct graph G
searchOrder $\leftarrow$ G.sortByDegree()
bundles $\leftarrow$ {}, bundlesConflict $\leftarrow$ {}
for i ***in*** *searchOrder* **do**
    needNew $\leftarrow$ True
    for $j = 1$ ***to*** *len(bundles)* **do**
        cnt $\leftarrow$ ConflictCnt(bundles[j],F[i])
        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: $newBin$, $binRanges$

Merge Exclusive Features

For the second issues, we need a good way of merging the features in the same bundle in order to reduce the corresponding training complexity. **The key is to ensure that the values of the original features can be identified from the feature bundles.** Since the histogram-based algorithm stores discrete bins instead of continuous values of the features, we can construct a feature bundle by letting exclusive features reside in different bins. This can be done by adding offsets to the original values of the features.



Experiments

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

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**

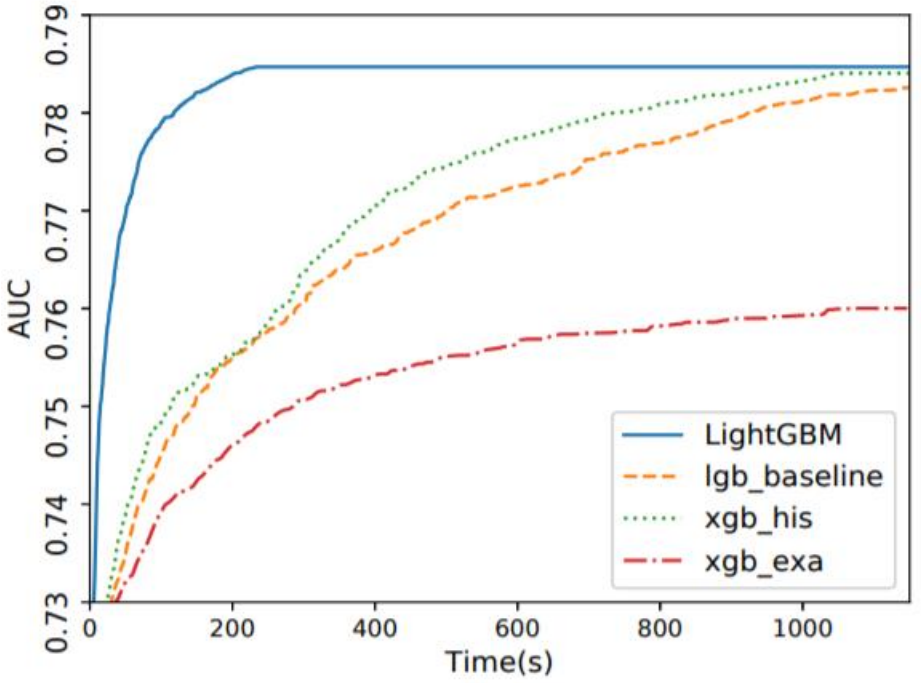


Figure 1: Time-AUC curve on Flight Delay.

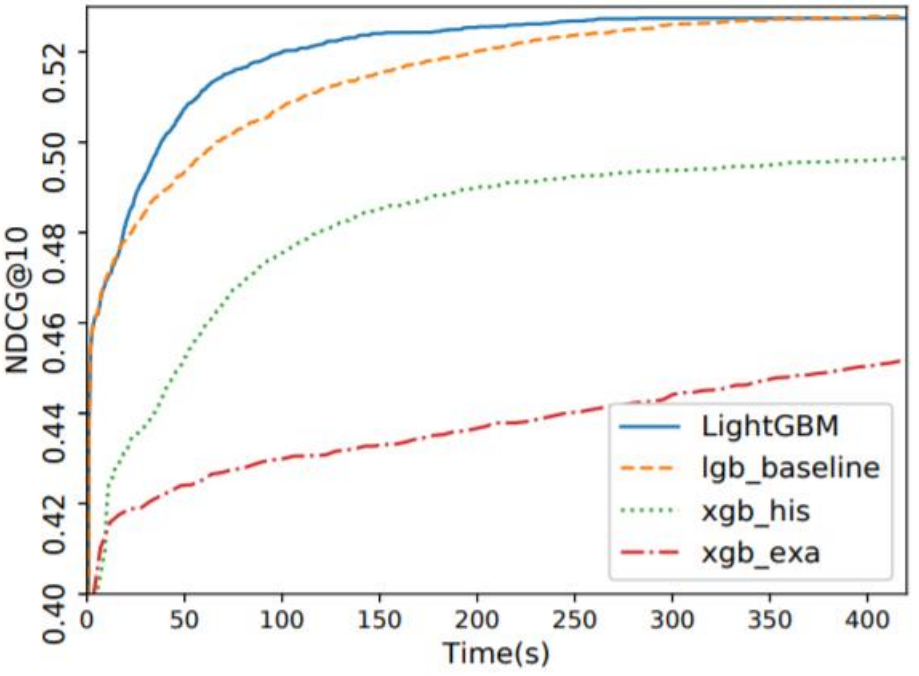


Figure 2: Time-NDCG curve on LETOR.