GBDT-MO: Gradient-Boosted Decision Trees for Multiple Outputs

Zhendong Zhang, *Graduate Student Member, IEEE*, and Cheolkon Jung, *Member, IEEE*

***Abstract*— Gradient-boosted decision trees (GBDTs) are widely used in machine learning, and the output of current GBDT implementations is a single variable. When there are multiple outputs, GBDT constructs multiple trees corresponding to the output variables. The correlations between variables are ignored by such a strategy causing redundancy of the learned tree structures. In this article, we propose a general method to learn GBDT for multiple outputs, called GBDT-MO. Each leaf of GBDT-MO constructs predictions of all variables or a subset of automatically selected variables. This is achieved by considering the summation of objective gains over all output variables. Moreover, we extend histogram approximation into the multiple-output case to speed up training. Various experiments on synthetic and real data sets verify that GBDT-MO achieves outstanding performance in terms of accuracy, training speed, and inference speed.**

***Index Terms*— Decision tree, gradient boosting, indirect regularization, multiple outputs, variable correlations.**

I. Introduction

MACHINE learning and data-driven approaches have achieved great success in recent years. Gradient-boosted decision tree (GBDT) [1], [2] is a powerful machine learning tool widely used in many applications, including multiclass classification [3], flocculation process modeling [4], learning to rank [5], learning to transfer [6], and click prediction [7]. It also produces state-of-the-art results for many data mining competitions, such as the Netflix prize [8]. GBDT uses decision trees as the base learner and sums the predictions of a series of trees. At each step, a new decision tree is trained to fit the residual between ground truth and current prediction. GBDT is popular due to its accuracy, efficiency, and interpretability. Many improvements have been proposed after [1]. XGBoost [9] used the second-order gradient to guide the boosting process and improve the accuracy. LightGBM [10] aggregated gradient information in histograms and significantly improved the training efficiency. CatBoost [11] proposed a novel strategy to deal with categorical features.

A limitation of current GBDT implementations is that the output of each decision tree is a single variable. This is because each leaf of a decision tree produces a single variable. However, multiple outputs are required for many machine learning problems, including, but not limited to, multiclass classification, multilabel classification [12], and multioutput regression [13]. Other machine learning methods, such as neural networks [14], can adapt to any dimension of outputs straightforwardly by changing the number of neurons in the last layer. The flexibility for the output dimension may be one of the reasons why neural networks are popular. However, since current GBDT methods are designed for single output, they are able to handle multiple outputs in isolation. At each step, they construct multiple decision trees, each of which corresponds to an individual output variable and then concatenates the predictions of all trees to obtain multiple outputs. This strategy is used in the most popular open-source GBDT library [9]–[11].

The major drawback of the abovementioned strategy is that correlations between variables are ignored during training because those variables are learned independently. However, correlations more or less exist between output variables. For example, it is proven that correlations between classes for multiclass classification improve the generalization ability of neural networks [15]. Ignoring variable correlations also leads to redundancy of the learned tree structures. Thus, it is necessary to learn GBDT for multiple outputs via better strategies. Up until now, a few works have explored it. Geurts *et al.* [16] transformed the multiple-output problem into a single-output problem by kernelizing the output space. However, this method was not scalable because the space complexity of its kernel matrix was n^2, where n is the number of training samples. Si *et al.* [17] proposed GBDT for sparse output (GBDT-sparse). They mainly focused on extreme multilabel classification problems, and the outputs were represented in a sparse format. A sparse split finding algorithm was designed for square hinge loss. Geurts *et al.* [16] and Si *et al.* [17] worked for specific loss, and they did not employ the second-order gradient and histogram approximation.

In this article, we propose a novel and general method to learn GBDT for multiple outputs, which is scalable and efficient, named GBDT-MO. Unlike previous works, we employ the second-order gradient and histogram approximation to improve GBDT-MO. The learning mechanism is designed based on them to jointly fit all variables in a single tree. Each leaf of a decision tree constructs multiple outputs at once. This is achieved by maximizing the summation of objective gains over all output variables. Sometimes, only a subset of the

Manuscript received September 2, 2019; revised December 27, 2019 and April 30, 2020; accepted July 9, 2020. Date of publication August 4, 2020; date of current version July 7, 2021. This work was supported in part by the National Natural Science Foundation of China under Grant 61872280 and in part by the International S&T Cooperation Program of China under Grant 2014DFG12780. *(Corresponding author: Cheolkon Jung.)*

The authors are with the School of Electronic Engineering, Xidian University, Xi'an 710071, China (e-mail: zhd.zhang.ai@gmail.com; ckjung@skku.edu).

Color versions of one or more of the figures in this article are available online at https://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNNLS.2020.3009776

output variables is correlated. It is expected that the proposed method automatically selects those variables and constructs predictions for them. We achieve this by adding L_0 constraint to the objective function. Since the learning mechanism of GBDT-MO enforces the learned trees to capture variable correlations, it plays a role in indirect regularization. Experiments on both synthetic and real data sets show that GBDT-MO achieves better generalization ability than the standard GBDT. Moreover, GBDT-MO achieves faster training and inference speeds,[1] especially when the number of outputs is large.

Compared with the existing methods, the main contributions of this article are as follows.

1) We formulate the problem of learning multiple outputs for GBDT and propose a split finding algorithm by deriving a general approximate objective for this problem.
2) To learn a subset of outputs, we add a sparse constraint to the objective. Based on it, we develop two sparse split finding algorithms.
3) We extend histogram approximation [18] into multiple-output case to speed up training.

The rest of this article is organized as follows. First, we review GBDT for single output and introduce basic definitions in Section II. Then, we describe the details of GBDT-MO in Section III. We address the related work in Section IV. Finally, we perform experiments and conclude in Sections V and VI, respectively.

II. GBDT for Single Output

In this section, we review GBDT for a single output. First, we describe the workflow of GBDT. Then, we show how to derive the objective of GBDT based on the second-order Taylor expansion of the loss, which is used in XGBoost. The objective of multiple-variable cases will be generalized in Section III. Finally, we explain the split finding algorithms that exactly or approximately minimize the objective.

A. Work Flow

Denote $\mathcal{D} = \{(\mathbf{x}_i, y_i)_{i=1}^n\}$ as a data set with n samples, where $\mathbf{x} \in \mathbb{R}^m$ is an m-dimensional input. Denote $f : \mathbb{R}^m \to \mathbb{R}$ as the function of a decision tree that maps $\mathbf{x}$ into a scalar. Since GBDT integrates t decision trees in an additive manner, the prediction of GBDT is $\hat{y}_i = \sum_{k=1}^t f_k(\mathbf{x}_i)$, where f_k is the function of kth decision tree. GBDT aims at constructing a series of trees given $\mathcal{D}$. It first calculates gradient based on current prediction at each boosting round and then constructs a new tree guided by gradient. Finally, it updates the prediction using the new tree. The most important part of GBDT is to construct trees based on gradient.

B. Objective

Based on the construction mechanism of decision trees, f can be further expressed as follows:

$$f(\mathbf{x}) = \mathbf{w}_{q(\mathbf{x})}, \quad q : \mathbb{R}^m \to [1, L], \quad \mathbf{w} \in \mathbb{R}^L \tag{1}$$

where L is the number of leaves of a decision tree, q is a function that selects a leaf given $\mathbf{x}$, and $\mathbf{w}_i$ is the value of the ith leaf. That is, once a decision tree is constructed, it first maps input into a leaf and then returns the value of that leaf.

Now, we consider the objective of the $(t+1)$th decision tree given prediction $\hat{y}$ of the first t trees as follows:

$$\sum_{i=1}^n l(\hat{y}_i + f(\mathbf{x}_i), y_i) + \lambda \mathcal{R}(f) \tag{2}$$

where the first term is fidelity term, and $\mathcal{R}$ is regularization term of f. In this work, λ is a positive number to control the tradeoff between fidelity term and regularization term. We suppose that l is a second-order differentiable loss. Based on the space of f, i.e., a constant value for each leaf, the fidelity term of (2) is separable with respect to each leaf. Then, (2) is rewritten as follows:

$$\sum_{j=1}^L \left\{ \sum_{i \in \text{leaf}_j} l(\hat{y}_i + \mathbf{w}_j, y_i) \right\} + \lambda \mathcal{R}(\mathbf{w}). \tag{3}$$

Although there are many choices of $\mathcal{R}$, we set $\mathcal{R}(\mathbf{w}) = (1/2)\|\mathbf{w}\|_2^2$, which is commonly used. Because (3) is separable with respect to each leaf, we only consider the objective of a single leaf as follows:

$$\mathcal{L} = \sum_i l(\hat{y}_i + w, y_i) + \frac{\lambda}{2} w^2 \tag{4}$$

where w is the value of a leaf and i is enumerated over the samples belonging to that leaf. $l(\hat{y}_i + w, y_i)$ can be approximated by the second-order Taylor expansion of $l(\hat{y}_i, y_i)$. Then, we have

$$\mathcal{L} = \sum_i \left\{ l(\hat{y}_i, y_i) + g_i w + \frac{1}{2} h_i w^2 \right\} + \frac{\lambda}{2} w^2 \tag{5}$$

where g_i and h_i are the first- and second-order derivatives of $l(\hat{y}_i, y)$ with respect to $\hat{y}_i$. By setting $(\partial \mathcal{L}/\partial w)$ to 0, we obtain the optimal value of w as follows:

$$w^* = -\frac{\sum_i g_i}{\sum_i h_i + \lambda}. \tag{6}$$

Substituting (6) into (5), we get the optimal objective as follows:

$$\mathcal{L}^* = -\frac{1}{2} \frac{\left(\sum_i g_i\right)^2}{\sum_i h_i + \lambda}. \tag{7}$$

We ignore $l(\hat{y}, y)$ since it is a constant term given $\hat{y}$.

C. Split Finding

One of the most important problems in decision tree learning is to find the best split given a set of samples. Especially, samples are divided into left and right parts based on the following rule:

$$\mathbf{x}_i \in \begin{cases} \text{left}, & \mathbf{x}_{ij} \le T \\ \text{right}, & \mathbf{x}_{ij} > T \end{cases} \tag{8}$$

where $\mathbf{x}_{ij}$ is the jth element of $\mathbf{x}_i$ and T is the threshold. The goal of split finding algorithms is to find the best column j and threshold T such that the gain between the optimal objectives before split and after split is maximized. The optimal objective

[1] In this article, training and inference speeds are averaged by boost rounds.

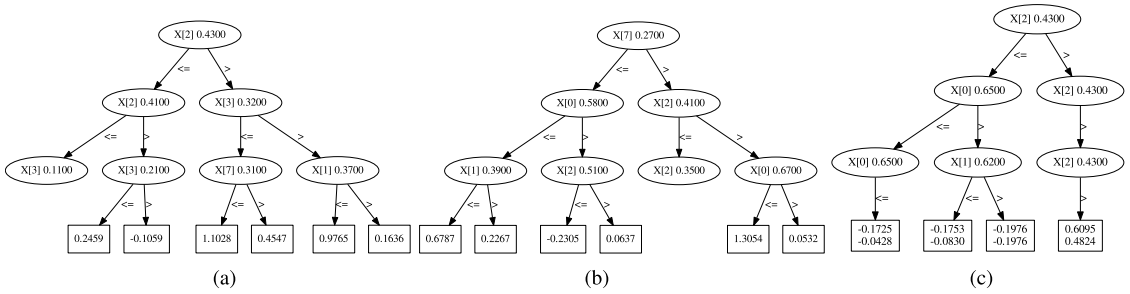

Fig. 1. Illustration of the learned trees in the first round on Yeast. The first and second columns show GBDT-SO for the first and second output variables, respectively. The last column shows GBDT-MO for the corresponding variables. We limit tree depth to 3 for illustration. (a) Variable 1. (b) Variable 2. (c) Variables 1 and 2.

after split is defined as the sum of the optimal objectives of left and right parts

$$\text{gain} = \mathcal{L}^* - \left(\mathcal{L}^*_{\text{left}} + \mathcal{L}^*_{\text{right}}\right) \tag{9}$$

where the maximizing gain is equivalent of minimizing $\mathcal{L}^*_{\text{left}} + \mathcal{L}^*_{\text{right}}$ because $\mathcal{L}^*$ is fixed for a given set of samples. The gain is used to determine whether a tree is grown. If the gain is smaller than a threshold, we stop the growth to avoid overfitting.

Exact and approximate split finding algorithms have been developed. The exact algorithm enumerates over all possible splits on all columns. For efficiency, it first sorts the samples according to the values of each column and then visits the samples in the sorted order to accumulate g and h that are used to compute the optimal objective. The exact split finding algorithm is accurate. However, when the number of samples is large, it is time-consuming to enumerate over all of the possible splits. Approximate algorithms are necessary as the number of samples increases. The key idea of approximate algorithms is that they divide samples into buckets and enumerate over these buckets instead of individual samples. The gradient statistics are accumulated within each bucket. The complexity of the enumeration process for split is independent of the number of samples. In the literature, there are two strategies for bucketing: quantile-based and histogram-based. Since the latter is significantly faster than the former [10], we focus on the latter in this work. For histogram-based bucketing, a bucket is called a bin. Samples are divided into b bins by b adjacent intervals: $(s_0, s_1, s_2, \ldots, s_b)$, where s_0 and s_b are usually set to $-\infty$ and $+\infty$, respectively. These intervals are constructed based on the distribution of the jth input column of the whole data set. Once constructed, they are keeping unchanged. Given a sample $\mathbf{x}_i$, it belongs to the kth bin if and only if $s_{k-1} < \mathbf{x}_{ij} <= s_k$. The bin value of $\mathbf{x}_{ij}$ is obtained by binary-search with complexity $\mathcal{O}(\log b)$. Given bin values, the histogram of the jth input column is constructed by a single fast scanning.

III. GBDT for Multiple Outputs

In this section, we describe GBDT-MO in detail. We first formulate the general problem of learning GBDT for multiple outputs. Especially, we derive the objective for learning multiple outputs based on the second-order Taylor expansion of loss. We approximate this objective and connect it with the objective for a single output. We also formulate the problem of learning a subset of variables and derive its objective. This is achieved by adding L_0 constraints. Then, we propose split finding algorithms that minimize the corresponding objectives. Finally, we discuss our implementations and analyze the complexity of our proposed split finding algorithms. Fig. 1 shows the difference of learned trees between GBDT-SO and GBDT-MO. In this work, we denote $\mathbf{X}_i$ is the ith row of a matrix $\mathbf{X}$, $\mathbf{X}_j$ is the jth column, and $\mathbf{X}_{ij}$ is its element of the ith row and the jth column.

A. Objective

We derive the objective of GBDT-MO. Each leaf of a decision tree constructs multiple outputs. Denote $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i)_{i=1}^n\}$ as a data set with n samples, where $\mathbf{x} \in \mathbb{R}^m$ is an m-dimensional input and $\mathbf{y} \in \mathbb{R}^d$ is a d-dimensional output instead of a scalar. Denote $f : \mathbb{R}^m \to \mathbb{R}^d$ as the function of a decision tree that maps $\mathbf{x}$ into the output space. Based on the construction mechanism of decision trees, f can be further expressed as follows:

$$f(\mathbf{x}) = \mathbf{W}_{q(\mathbf{x})}, \quad q : \mathbb{R}^m \to [1, L], \quad \mathbf{W} \in \mathbb{R}^{L \times d} \tag{10}$$

where L is the number of leaves of a decision tree, q is a function that selects a leaf given $\mathbf{x}$, and $\mathbf{W}_i \in \mathbb{R}^d$ is the values of the ith leaf. That is, once a decision tree is constructed, it first maps an input into a leaf and then returns the d-dimensional vector of that leaf. Next, the prediction of the first t trees is $\hat{\mathbf{y}}_i = \sum_{k=1}^t f(\mathbf{x}_i)$.

We consider the objective of the $(t+1)$th tree given $\hat{\mathbf{y}}$. Because it is separable with respect to each leaf [see (2) and (3)], we only consider the objective of a single leaf as follows:

$$\mathcal{L} = \sum_i l(\hat{\mathbf{y}}_i + \mathbf{w}, \mathbf{y}_i) + \lambda \mathcal{R}(\mathbf{w}). \tag{11}$$

We highlight that $\mathbf{w} \in \mathbb{R}^d$ is a vector with d elements that belong to a leaf. Again, we suppose that l is a second-order differentiable function. $l(\hat{\mathbf{y}}_i + \mathbf{w}, \mathbf{y}_i)$ can be approximated by the second-order Taylor expansion of $l(\hat{\mathbf{y}}_i, \mathbf{y}_i)$. Setting $\mathcal{R}(\mathbf{w}) = (1/2)\|\mathbf{w}\|_2^2$, we have

$$\mathcal{L} = \sum_i \left\{ l(\hat{\mathbf{y}}_i, \mathbf{y}_i) + (\mathbf{g})_i^T \mathbf{w} + \frac{1}{2}\mathbf{w}^T (\mathbf{H})_i \mathbf{w} \right\} + \frac{\lambda}{2}\|\mathbf{w}\|_2^2 \tag{12}$$

where $(\mathbf{g})_i = (\partial l/\partial \hat{\mathbf{y}}_i)$ and $(\mathbf{H})_i = (\partial^2 l/\partial \hat{\mathbf{y}}_i^2)$. To avoid notation conflicts with the subscript of vectors or matrices, we use $(\cdot)_i$ to indicate that an object belongs to the ith sample. This notation is omitted when there is no ambiguity. By setting $(\partial \mathcal{L}/\partial \mathbf{w}) = \mathbf{0}$ for (12), we obtain the optimal leaf values

$$\mathbf{w}^* = -\left(\sum_i (\mathbf{H})_i + \lambda \mathbf{I}\right)^{-1} \left(\sum_i (\mathbf{g})_i\right) \tag{13}$$

where $\mathbf{I}$ is an identity matrix. By substituting $\mathbf{w}^*$ into (12) and ignoring the constant term $l(\hat{\mathbf{y}}_i, \mathbf{y}_i)$, we get the optimal objective as follows:

$$\mathcal{L}^* = -\frac{1}{2}\left(\sum_i (\mathbf{g})_i\right)^T \left(\sum_i (\mathbf{H})_i + \lambda \mathbf{I}\right)^{-1} \left(\sum_i (\mathbf{g})_i\right). \tag{14}$$

We have derived the optimal leaf values and the optimal objective for multiple outputs. Comparing (13) with (6) and (14) with (7), it is easy to see that this is a natural generalization of the single-output case. In fact, when the loss function l is separable with respect to different output dimensions, or equivalently, when its hessian matrix $\mathbf{H}$ is diagonal, each element of $\mathbf{w}^*$ is obtained by the same way as in (6)

$$\widetilde{\mathbf{w}}_j^* = -\frac{\sum_i (\mathbf{g}_j)_i}{\sum_i (\mathbf{h}_j)_i + \lambda} \tag{15}$$

where $\mathbf{h} \in \mathbb{R}^d$ is the diagonal elements of $\mathbf{H}$. The optimal objective in (14) can be expressed as the sum of objectives over all the output dimensions

$$\widetilde{\mathcal{L}}^* = -\frac{1}{2}\sum_{j=1}^{d}\left\{\frac{\left(\sum_i (\mathbf{g}_j)_i\right)^2}{\sum_i (\mathbf{h}_j)_i + \lambda}\right\}. \tag{16}$$

GBDT-MO and GBDT are different even when $\mathbf{H}$ is diagonal because GBDT-MO considers the objectives of all output variables at the same time.

However, it is problematic when l is not separable, or equivalently, $\mathbf{H}$ is nondiagonal. First, it is difficult to store $\mathbf{H}$ for every sample when the output dimension d is large. Second, to get the optimal objective for each possible split, it is required to compute the inverse of a $d \times d$ matrix, which is time-consuming. Thus, it is impractical to learn GBDT-MO using the exact objective in (14) and the exact leaf values in (13). Fortunately, it is shown in [19] that $\widetilde{\mathbf{w}}^*$ and $\widetilde{\mathcal{L}}^*$ are good approximations of the exact leaf values and the exact objective when the diagonal elements of $\mathbf{H}$ are dominated. $\widetilde{\mathbf{w}}^*$ and $\widetilde{\mathcal{L}}^*$ are derived from an upper bound of $l(\hat{\mathbf{y}}, \mathbf{y})$. See Appendix A for details and further discussion. Although it is possible to derive better approximations for some specific loss functions, we use the abovementioned diagonal approximation in this work. We leave better approximations as our future work.

B. Sparse Objective

In Section III-A, all output variables are involved because there are correlations among variables. However, only a subset of variables is correlated in practice. Thus, it is required to learn the values of a suitable subset of $\mathbf{w}$ in a leaf. Although only a subset of variables is covered in a leaf, all variables can be covered because there are many leaves. Moreover, sparse objective is able to reduce the number of parameters. We obtain sparse objective by adding L_0 constraint to the nonsparse one. Based on the diagonal approximation, the optimal sparse leaf values are as follows:

$$\begin{aligned}\mathbf{w}_{\text{sp}}^* = \arg\min_{\mathbf{w}} \; & G^T\mathbf{w} + \frac{1}{2}\mathbf{w}^T(\text{diag}(H) + \lambda \mathbf{I})\mathbf{w} \\ \text{s.t. } & \|\mathbf{w}\|_0 <= k\end{aligned} \tag{17}$$

where k is the maximum nonzero elements of $\mathbf{w}$. We denote $G = \sum_i (\mathbf{g})_i$ and $H = \sum_i (\mathbf{h})_i$ to simplify the notation. Note the difference between H and $\mathbf{H}$.

For the jth element of $\mathbf{w}_{\text{sp}}^*$, its value is either $-(G_j/(H_j + \lambda))$ or 0. Accordingly, the objective contributed by the jth column is either $-(1/2)(G_j^2/(H_j + \lambda))$ or 0. Thus, to minimize (17), we select k columns with the largest $\upsilon_j = (G_j^2/(H_j + \lambda))$. Let π be the sorted order of υ such that

$$\upsilon_{\pi^{-1}(1)} \geq \upsilon_{\pi^{-1}(2)} \geq \cdots \geq \upsilon_{\pi^{-1}(d)}. \tag{18}$$

Then, the solution of (17) is as follows:

$$\left(\mathbf{w}_{\text{sp}}^*\right)_j = \begin{cases} -\dfrac{G_j}{H_j + \lambda}, & \pi(j) \leq k \\ 0, & \pi(j) > k. \end{cases} \tag{19}$$

That is, columns with k largest υ keep their values, while others set to 0. The corresponding optimal objective is as follows:

$$\mathcal{L}_{\text{sp}}^* = -\frac{1}{2}\sum_{j:\pi(j)\leq k} \frac{G_j^2}{H_j + \lambda}. \tag{20}$$

Our sparse objective is similar to the objective in [17]. Since GBDT-sparse only uses the first order derivative, the second-order derivative is set to a constant (1.0) for GBDT-sparse. Thus, H_j in (20) is the number of samples belonging to the leaf. The final solution of the sparse objective only contains a subset of variables. However, to get the optimal subset, all variables are involved in comparisons. That is, all variables are indirectly used.

C. Split Finding

Split finding algorithms for single output maximize the gain of objective before and after the split. When dealing with multiple outputs, the objective is defined over all the output variables. To find the maximum gain, it is required to scan all columns of outputs. The exact algorithm is inefficient because it enumerates all possible splits. In this work, we use the histogram approximation-based one to speed up the training process. To deal with multiple outputs, one should extend the histogram construction algorithm from a single-variable case into a multiple-variable case. Such an extension is straightforward. Denote b as the number of bins of a histogram. Then, the gradient information is stored in a $b \times d$ matrix, and its jth column corresponds to the gradient information of the jth variable of outputs. We describe the histogram construction algorithm for multiple outputs in Algorithm 1.

Algorithm 1 Histogram for Multiple Outputs

Input: the set of used samples $\mathcal{S}$, column index k, output dimension d and number of bins b
Output: histogram of kth column
Initialize Hist.$\mathbf{g} \in \mathbb{R}^{b\times d}$, Hist.$\mathbf{h} \in \mathbb{R}^{b\times d}$ and Hist.$\mathbf{cnt} \in \mathbb{R}^{b}$
for i **in** $\mathcal{S}$ **do**
  $bin \leftarrow$ bin value of $\mathbf{x}_{ik}$
  Hist.$\mathbf{cnt}[bin] \leftarrow$ Hist.$\mathbf{cnt}[bin] + 1$
  for $j = 1$ **to** d **do**
    Hist.$\mathbf{g}[bin][j] \leftarrow$ Hist.$\mathbf{g}[bin][j] + (\mathbf{g}_j)_i$
    Hist.$\mathbf{h}[bin][j] \leftarrow$ Hist.$\mathbf{h}[bin][j] + (\mathbf{h}_j)_i$
  end for
end for

Algorithm 2 Approximate Split Finding for Multiple Outputs

Input: histograms of current node, input dimension m and output dimension d
Output: split with maximum $gain$
$gain \leftarrow 0$
for $k = 1$ **to** m **do**
  Hist $\leftarrow$ histogram of kth column
  $b \leftarrow$ number of bins of Hist
  $G \leftarrow \sum_{i=1}^{b}$ Hist.$\mathbf{g}[i]$, $H \leftarrow \sum_{i=1}^{b}$ Hist.$\mathbf{h}[i]$
  $G^l \leftarrow \mathbf{0}$, $H^l \leftarrow \mathbf{0}$
  for $i = 1$ **to** b **do**
    $G^l \leftarrow G^l +$ Hist.$\mathbf{g}[i]$, $H^l \leftarrow H^l +$ Hist.$\mathbf{h}[i]$
    $G^r \leftarrow G - G^l$, $H^r \leftarrow H - H^l$
    $score \leftarrow \sum_{j=1}^{d} \left\{ \frac{(G_j^l)^2}{H_j^l+\lambda} + \frac{(G_j^r)^2}{H_j^r+\lambda} - \frac{(G_j)^2}{H_j+\lambda} \right\}$, (16)
    $gain \leftarrow \max(gain, score)$
  end for
end for

Once the histogram is constructed, we scan its bins to find the best split. We describe this histogram-based split finding algorithm in Algorithm 2. Compared with the single-output one, the objective gain is the sum of gains over all outputs.

D. Sparse Split Finding

We propose histogram approximation-based sparse split finding algorithms. Compared with the nonsparse one, the key difference is to compute their objective gain given a possible split as follows:

$$\frac{1}{2}\left\{ \sum_{i:\pi^l(i)\le k} \frac{\left(G_i^l\right)^2}{H_i^l+\lambda} + \sum_{j:\pi^r(j)\le k} \frac{\left(G_j^r\right)^2}{H_j^r+\lambda} \right\} - \text{const} \quad (21)$$

where π^l is the sorted order of $((G^l)^2/(H^l+\lambda))$ and π^r is the sorted order of $((G^r)^2/(H^r+\lambda))$, respectively. const means that the objective before split is fixed for every possible split. When we scan over columns, we maintain the top-k columns for both parts whose $((G)^2/(H+\lambda))$ is the largest. This can be achieved by a top-k priority queue. We describe the algorithm in Algorithm 3.

In Algorithm 3, the sets of the selected columns of left and right parts are not completely overlapping. We restrict those two sets to be completely overlapping. In other word, the selected columns of two parts are shared. Then, the objective gain in such a case becomes

$$\frac{1}{2}\sum_{i:\pi(i)\le k}\left\{ \frac{\left(G_i^l\right)^2}{H_i^l+\lambda} + \frac{\left(G_i^r\right)^2}{H_i^r+\lambda} \right\} - \text{const} \quad (22)$$

where π is the sorted order of $((G^l)^2/(H^l+\lambda)) + ((G^r)^2/(H^r+\lambda))$. We call it the restricted sparse split finding algorithm. We describe the gain computing for it in Algorithm 4. There are two advantages of the restricted one, which are as follows.

1) It has lower computational complexity because it only maintains a single top-k priority queue.
2) It introduces smoothness prior into the function space because it makes two child nodes with the same parent more similar.

We provide an example to show the differences between nonsparse split finding, sparse split finding, and restricted sparse split finding in Fig. 2.

Algorithm 3 Gain for Sparse Split Finding

Input: gradient statistics of current split, output dimension d and sparse constraint k.
Output: gain of current split
$Q_l, Q_r \leftarrow$ top-k priority queue
for $j = 1$ **to** d **do**
  append $\frac{(G_j^l)^2}{H_j^l+\lambda}$ to Q_l, append $\frac{(G_j^r)^2}{H_j^r+\lambda}$ to Q_r
end for
$score = \sum_{v_l\in Q_l} v_l + \sum_{v_r\in Q_r} v_r$, (21)

Algorithm 4 Gain for Restricted Sparse Split Finding

Input: gradient statistics of current split, output dimension d and sparse constraint k.
Output: gain of current split
$Q \leftarrow$ top-k priority queue
for $j = 1$ **to** d **do**
  append $\frac{(G_j^l)^2}{H_j^l+\lambda} + \frac{(G_j^r)^2}{H_j^r+\lambda}$ to Q
end for
$score = \sum_{v\in Q} v$, (22)

E. Implementation Details

We implement GBDT-MO from scratch by C++. First, we implement the core function of LightGBM for ourselves, called GBDT-SO. Then, integrate the learning mechanism for multiple outputs into it, i.e., GBDT-MO.[2] We also provide a Python interface. We speed up GBDT-MO using multicore parallelism, implemented with OpenMP. A decision tree grows up in a best-first manner. Especially, we store the information of nodes that have not been divided into memory. At each time, when we need to add a node, we select the node whose objective gain is the maximum from all stored nodes.

[2]Our codes are available in https://github.com/zzd1992/GBDTMO

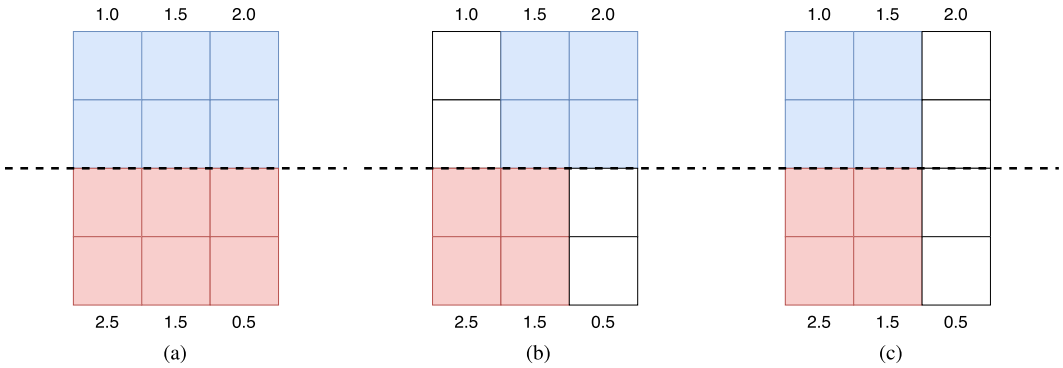

Fig. 2. Examples of different split algorithms for four samples in rows and three outputs in columns. The sparse constraint is set to 2. The dashed line divides samples into two parts. Numbers denote the negative objectives for each column. Selected columns of the left part are marked in blue and selected columns of the right part are marked in red. Best viewed on the screen. (a) Nonsparse split finding. (b) Sparse split finding. (c) Restricted sparse split finding.

Algorithm 5 Tree Growth

Input: set of samples $\mathcal{S}$, gradient statistics, and hyper-parameters for tree learning.
Output: a decision tree
$Hist \leftarrow$ Histograms of $\mathcal{S}$
$Split \leftarrow SplitFinding(Hist)$
$Q \leftarrow$ priority queue sorted by $Split.gain$
$Q.push(Split, Hist, \mathcal{S})$
repeat
  $Split, Hist, \mathcal{S} \leftarrow Q.pop()$
  $Hist^l, Hist^r, \mathcal{S}^l, \mathcal{S}^r \leftarrow ApplySplit(Split, Hist, \mathcal{S})$
  if stop condition is not meet **then**
    $Split^l \leftarrow SplitFinding(Hist^l)$
    $Q.push(Split^l, Hist^l, \mathcal{S}^l)$
  end if
  if stop condition is not meet **then**
    $Split^r \leftarrow SplitFinding(Hist^r)$
    $Q.push(Split^r, Hist^r, \mathcal{S}^r)$
  end if
until Q is empty

In practice, we store up to 48 nodes in memory to reduce the memory cost. We describe the algorithm for the growth of a tree in Algorithm 5.

F. Complexity Analysis

We analyze the training and inference complexity for GBDT-SO and GBDT-MO. For the training complexity, we focus on the complexity of split finding algorithms. Recall that the input dimension is m, the output dimension is d, the sparse constraint is k, the number of bins is b, the number of boosting rounds is t, and the maximum tree depth is h. Suppose that b is fixed for all input dimensions. Table I shows the training complexity. The complexity of nonsparse split finding is $\mathcal{O}(bmd)$ because it enumerates on histogram bins, input dimensions, and output dimensions. For the sparse case, it is multiplied by $\log k$ because inserting an element into a top-k priority queue requires $\mathcal{O}(\log k)$ comparisons (see Algorithm 4). When the exact Hessian is used [see (14)], it becomes $\mathcal{O}(bmd^3)$ because calculating the inverse of a $d \times d$ matrix requires $\mathcal{O}(d^3)$ operations. GBDT-SO has the same complexity as GBDT-MO for split finding. However, it does not mean that GBDT-SO is as fast as GBDT-MO. Beyond split finding, GBDT-SO has high computational overhead that slows down its training speed. Table II shows the inference complexity. It can be observed that GBDT-SO needs d times more comparisons than GBDT-MO.

TABLE I
COMPLEXITY OF SPLIT FINDING

Methods	Complexity
GBDT-SO	$\mathcal{O}(bmd)$
GBDT-MO	$\mathcal{O}(bmd)$
GBDT-MO (sparse)	$\mathcal{O}(bmd \log k)$
GBDT-MO (exact hessian)	$\mathcal{O}(bmd^3)$

TABLE II
COMPLEXITY OF INFERENCE

Methods	Comparisons	Additions
GBDT-SO	$\mathcal{O}(tdh)$	$\mathcal{O}(td)$
GBDT-MO	$\mathcal{O}(th)$	$\mathcal{O}(td)$
GBDT-MO (sparse)	$\mathcal{O}(th)$	$\mathcal{O}(tk)$

For training, XGBoost and LightGBM have the same complexity as GBDT-SO. The complexity of GBDT-sparse is $\mathcal{O}(||G||_0 \log k)$, where $||G||_0$ is the number of nonzero gradient elements, which grows as the data set becomes large. For inference, XGBoost and LightGBM have the same complexity as GBDT-SO, while GBDT-sparse has the same complexity as GBDT-MO when a sparse split finding is used.

IV. RELATED WORK

GBDT proposed in [1] has two characteristics: 1) it uses decision trees as the base learner and 2) its boosting process is guided by the gradient of some loss function. Since then, many variants of [1] have been proposed. The second-order gradient is used in XGBoost [9] to guide its boost process that improves accuracy. The histogram approximation of split finding proposed in [18] is used as the base algorithm in LightGBM [10] to improve training efficiency. Those two improvements are used in the proposed GBDT-MO.

Since machine learning problems with multiple outputs become common, many tree- or boosting-based methods have been proposed to deal with multiple outputs. Agrawal *et al.* [20] and Prabhu and Varma [21] generalize the impurity measures defined for binary classification and ranking tasks to a multilabel scenario for splitting a node. However, they are random forest-based methods. That is, new trees are not constructed in a boosting manner. Several works extend adaptive boost (AdaBoost) into multilabel cases, such as AdaBoost.MH [22] and AdaBoost.LC [23]. The spirits of AdaBoost.MH and AdaBoost.LC are different from GBDT. At each step, a new base learner is trained from scratch on the reweighted samples. Moreover, AdaBoost.MH only works for Hamming loss, while AdaBoost.LC only works for the covering loss [23].

They do not belong to GBDT families. Two works that belong to GBDT families have been proposed for learning multiple outputs [16], [17]. Geurts *et al.* [16] transform the multiple-output problem into the single-output problem by kernelizing the output space. To achieve this, an $n \times n$ kernel matrix should be constructed where n is the number of training samples. Thus, this method is not scalable. Moreover, it works only for square loss. GBDT-sparse is proposed in [17]. The outputs are represented in a sparse format. A sparse split finding algorithm is designed by adding an L_0 constraint to the objective. The sparse split finding algorithms of GBDT-MO are inspired by this work. There are several differences between GBDT-MO and GBDT-sparse that are given as follows.

1) GBDT-sparse requires the loss to be separable over output variables. It also requires that its gradient is sparse, i.e., $(\partial l(\hat{y}, y)/\partial \hat{y}) = 0$ when $\hat{y} = y$. During training, it introduces a clipping operator into the loss to maintain the sparsity of gradient, and thus, limited types of loss are used.
2) GBDT-sparse does not employ the second-order gradient and histogram approximation. It is worthwhile because it is not clear how to construct sparse histograms, especially when combining with the second-order gradient.
3) The main motivation of GBDT-sparse is to reduce the space complexity of training and the size of models, whereas the main motivation of GBDT-MO is to improve its generalization ability by capturing correlations between the output variables.

It may be hard to store the outputs in memory without sparse format when the output dimension is very large. In such a situation, GBDT-sparse is a better choice.

V. Experiments

We evaluate GBDT-MO on problems of multioutput regression, multiclass classification, and multilabel classification. First, we show the benefits of GBDT-MO using two synthetic problems. Then, we evaluate GBDT-MO on six real data sets. We further evaluate our sparse split finding algorithms. Finally, we analyze the impact of diagonal approximation on the performance. Recall that GBDT-SO is our own implementation of GBDT for a single output. Except for the split finding algorithm, all implementation details are the same as GBDT-MO for a fair comparison. The purpose of our experiments is not pushing state-of-the-art results on specific data sets. Instead, we would show that GBDT-MO has better generalization ability than GBDT-SO. On synthetic data sets, we compare GBDT-MO with GBDT-SO. On real data sets, we compare GBDT-MO with GBDT-SO, XGBoost, LightGBM, and GBDT-sparse[3] in terms of accuracy, training speed, inference speed, and memory usage. XGBoost and LightGBM are state-of-the-art GBDT implementations for single output, while GBDT-sparse is the most relevant work to GBDT-MO. All experiments are conducted on a workstation with Intel Xeon CPU E5-2698 v4. We use four threads on synthetic data sets, while we use eight threads on real data sets. We provide hyperparameter settings in Appendix B. The training process is terminated when the performance does not improve within 25 rounds.

TABLE III
RMSE on Synthetic Data Sets

	friedman1	random projection
GBDT-SO	0.1540	0.0204
GBDT-MO	**0.1429**	**0.0180**

A. Synthetic Data Sets

The first data set is derived from the friedman1 regression problem [24]. Its target y is generated by

$$f(\mathbf{x}) = \sin(\pi \mathbf{x}_1 \mathbf{x}_2) + 2(\mathbf{x}_3 - 0.5)^2 + \mathbf{x}_4 + 0.5\mathbf{x}_5 \quad (23)$$

$$y = f(\mathbf{x}) + 0.1\varepsilon \quad (24)$$

where $\mathbf{x} \in \mathbb{R}^{10}$ and $\varepsilon \sim \mathcal{N}(0; 1)$. Each element of $\mathbf{x}$ is sampled from $\mathcal{U}(-1, 1)$. The last five elements of $\mathbf{x}$ are irrelevant to the target. We extend this problem into multiple-output case by adding independent noise to $f(\mathbf{x})$. That is, $\mathbf{y}_i = f(\mathbf{x}) + 0.1\varepsilon$, where $\mathbf{y} \in \mathbb{R}^5$.

The second data set is generated by random projection

$$\mathbf{y} = \mathbf{w}^T \mathbf{x} \quad (25)$$

where $\mathbf{x} \in \mathbb{R}^4$, $\mathbf{w} \in \mathbb{R}^{4\times 8}$, and $\mathbf{y} \in \mathbb{R}^8$. Each element of $\mathbf{x}$ and $\mathbf{w}$ is independently sampled from $\mathcal{U}(-1, 1)$.

For both data sets, we generate 10 000 samples for training and 10 000 samples for test. We train them via mean square error (MSE) and evaluate the performance of test samples via root-mean-square error (RMSE). We repeat the experiments five times with different seeds and average the RMSE. As shown in Table III, GBDT-MO is better than GBDT-SO. We provide the training curves in Fig. 3. For fairness, curves of different methods are plotted with the same learning rate and maximum tree depth. It can be observed that GBDT-MO has better generalization ability on both data sets. This is because its RMSE on test samples is lower, and its performance gap is smaller. Here, the performance gap means the differences in performance between training

[3]GBDT-sparse is implemented for ourselves, and we do not report its training time due to the slow speed.

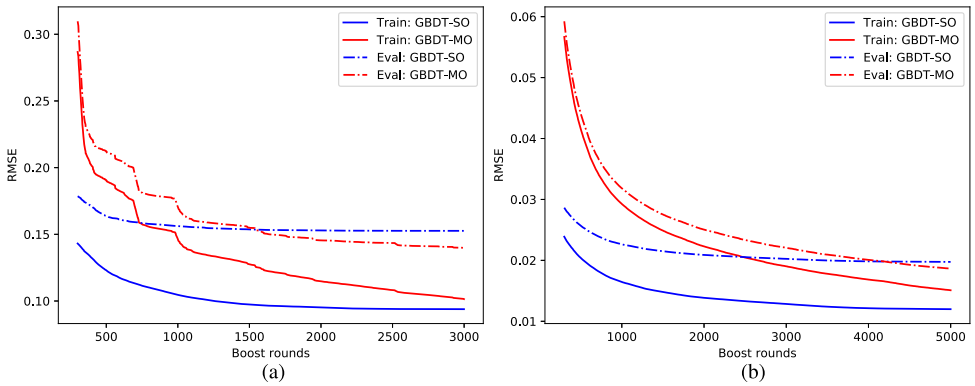

Fig. 3. RMSE curves on friedman1 and random project. RMSE curves for training samples are drawn in the solid lines, while RMSE curves for test samples are drawn in the dashed lines. Best viewed on the screen. (a) Friedman1. (b) Random projection.

TABLE IV
DATA SET STATISTICS. * MEANS THE FEATURES ARE PREPROCESSED

Dataset	# Training samples	# Test samples	# Features	# Outputs	Problem type
MNIST	50,000	10,000	784	10	classification
Yeast	1,038	446	8	10	classification
Caltech101	6,073	2,604	324*	101	classification
MNIST-inpainting	50,000	10,000	200*	24	regression
Student-por	454	159	41*	3	regression
NUS-WIDE	161,789	107,859	128*	81	multi-label

samples and unseen samples, which is usually used to measure the generalization ability of machine learning algorithms. The output variables of friedman1 are correlated because they are observations of the same underlying variable corrupted by the Gaussian noise. The output variables of random projection are also correlated because this projection is overcomplete. This supports our claim that GBDT-MO has better generalization abilities because its learning mechanism encourages it to capture variable correlations. However, GBDT-MO suffers from slow convergence speed.

B. Real Data Sets

In this section, we evaluate GBDT-MO on six real data sets and compare with related methods.

MNIST[4] is widely used for classification whose samples are 28×28 gray images of handwritten digits from 0 to 9. Each sample is converted into a vector with 784 elements.

Yeast[5] has eight input attributions, each of which is a measurement of the protein sequence. The goal is to predict protein localization sites with ten possible choices.

Caltech101[6] contains images of objects belonging to 101 categories. Most categories have about 50 samples. The size of each image is roughly 300×200. To obtain fixed-length features, we resize each image into 64×64 and compute the HOG descriptor [25] of the resized image. Each sample is finally converted to a vector with 324 elements.

MNIST-inpainting is a regression task based on MNIST. We crop the central 20×20 patch from the original 28×28 image because most boundary pixels are 0. Then, the cropped image is divided into upper and lower halves, each of which is a 10×20 patch. The upper half is used as the input. We further crop a 4×6 small patch at the top center of the lower half. This small patch is used as the target. The task is to predict the pixels of this 4×6 patch given the upper half image.

Student-por[7] predicts the Portuguese language scores in three different grades of students based on their demographic-, social-, and school-related features. The original scores range from 0 to 20. We linearly transform them into $[-1, 1]$. We use one-hot coding to deal with the categorical features of this data set.

NUS-WIDE[8] is a data set for real-world web image retrieval [26]. Tsoumakas *et al.* [12] select a subset label of this data set and uses it for multilabel classification. Images are represented using 128-D cVLAD+ features described in [27].

We summarize the statistics of the above data sets in Table IV. Those data sets are diverse in terms of scale and complexity. For MNIST, MNIST-inpainting, and NUS-WIDE, we use the official training-test split. For others, we randomly select 70% samples for training and the rest for the test. We repeat this strategy ten times with different seeds and report the average results. For multiclass classification, we use cross-entropy loss. For regression and multilabel classification, we use MSE loss. For regression, the performance is measured by RMSE. For multiclass classification and multilabel classification, the performance is measured by top-one accuracy.

1) Accuracy: RMSE and top-one accuracy on real data sets are shown in Table V. The overall performance of GBDT-MO

[4] yann.lecun.com/exdb/mnist/
[5] archive.ics.uci.edu/ml/datasets/Yeast
[6] www.vision.caltech.edu/Image_Datasets/Caltech101/
[7] archive.ics.uci.edu/ml/datasets/Student+Performance
[8] mulan.sourceforge.net/datasets-mlc.html

TABLE V
PERFORMANCE ON REAL DATA SETS

	MNIST	Yeast	Caltech101	NUS-WIDE	MNIST-inpaining	Student-por
	accuracy	accuracy	accuracy	top-1 accuracy	RMSE	RMSE
XGBoost	97.86	**62.94**	56.52	43.72	0.26088	0.24623
LightGBM	98.03	61.97	55.94	43.99	0.26090	0.24466
GBDT-sparse	96.41	62.83	43.93	44.05	-	-
GBDT-SO	98.08	61.97	56.62	44.10	0.26157	0.24408
GBDT-MO	**98.30**	62.29	**57.49**	**44.21**	**0.26025**	**0.24392**

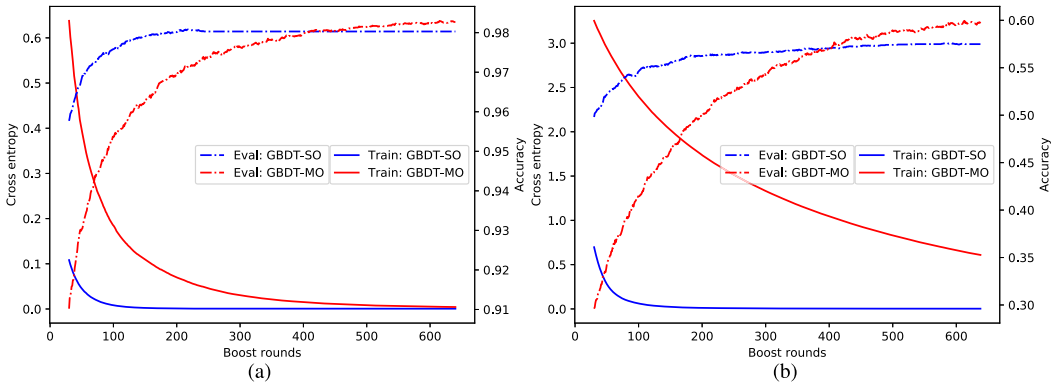

Fig. 4. Cross entropy curves and accuracy curves on MNIST and Caltech101. Cross entropy curves for training samples are drawn in the solid lines, while accuracy curves for test samples are drawn in the dashed lines. Best viewed on the screen. (a) MNIST. (b) Caltech101.

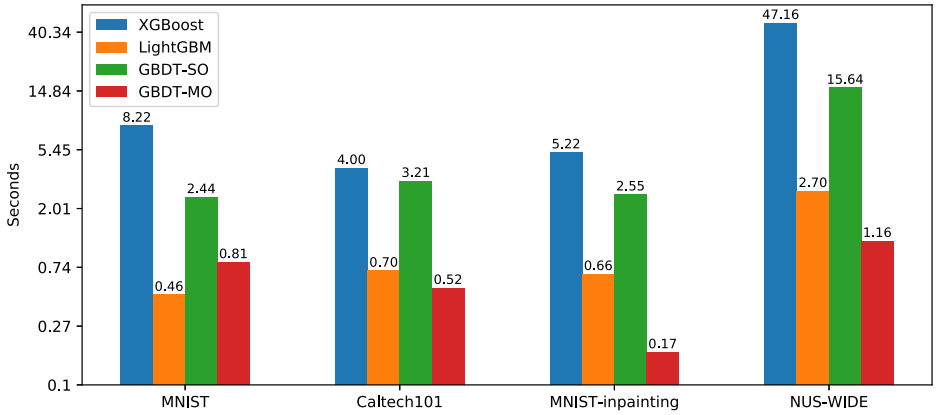

Fig. 5. Average training time for each boost round (unit: second). Four relatively larger data sets are used. Y-axis is plotted in the log scale.

TABLE VI
AVERAGED INFERENCE TIME FOR EACH ROUND (UNIT: MILLISECOND)

	MNIST	Caltech101	NUS-WIDE	MNIST-inpaining
XGBoost	6.38	9.57	119.24	6.61
LightGBM	1.78	2.76	209.79	8.50
GBDT-sparse	**0.57**	0.29	**5.37**	-
GBDT-SO	3.77	4.70	515.90	12.75
GBDT-MO	0.59	**0.15**	7.38	**0.32**

TABLE VII
MEMORY USAGE OF DIFFERENT METHODS (UNIT: MEGABIT)

	MNIST	Caltech101	NUS-WIDE	MNIST-inpaining
XGBoost	1024	315	683	443
LightGBM	563	788	442	145
GBDT-sparse	659	61	504	-
GBDT-SO	**193**	**34**	**128**	**67**
GBDT-MO	210	121	199	89

is better than others. We provide statistical tests in Appendix C to show the performance of GBDT-MO on data sets with random training and testing split. We compare the loss curves and the accuracy curves of GBDT-MO and GBDT-SO on MNIST and Caltech101 in Fig. 4. It can be concluded that GBDT-MO has better generalization ability than GBDT-SO. This is because its training loss is higher, while its test performance is better.

2) Training Time: We run ten boost rounds and record the average training time for each boost round. We repeat this process three times and report the average training time in seconds, as shown in Fig. 5. The training speed for Yeast and Student-por is not reported because those two data sets are too small. GBDT-MO is remarkably faster than GBDT-SO and XGBoost, especially when the number of outputs is large. The training time means the time for a single round

TABLE VIII
TRAINING TIME FOR SPARSE SPLIT FINDING

Restricted/Unrestricted	MNIST	MNIST-inpaining	Caltech101	NUS-WIDE
$k = 2$	0.619/**0.599**	-	-	-
$k = 4$	**0.737**/0.758	**0.149**/0.179	-	-
$k = 8$	**0.721**/0.778	**0.161**/0.195	**0.578**/0.663	**1.248**/1.266
$k = 16$	-	**0.153**/0.208	**0.620**/0.738	**1.237**/1.365
$k = 32$	-	-	**0.628**/0.866	**1.256**/1.453
$k = 64$	-	-	**0.650**/0.937	**1.263**/1.493

TABLE IX
PERFORMANCE FOR SPARSE SPLIT FINDING

Restricted/Unrestricted	MNIST	MNIST-inpaining	Caltech101	NUS-WIDE
	accuracy	RMSE	accuracy	top-1 accuracy
$k = 2$	97.54/**97.79**	-	-	-
$k = 4$	**98.19**/98.18	0.26424/**0.26368**	-	-
$k = 8$	**98.07**/98.06	**0.26245**/0.26391	53.57/**54.11**	44.20/**44.20**
$k = 16$	-	**0.26232**/0.26248	**55.98**/54.69	44.22/**44.27**
$k = 32$	-	-	**56.71**/56.64	**44.07**/44.04
$k = 64$	-	-	**58.09**/57.58	**44.23**/44.22

because researchers care more about it. It is slightly faster than LightGBM. Since we eliminate the interference from outer factors, the comparisons between GBDT-SO and GBDT-MO demonstrate the effectiveness of the proposed method.

3) Inference Time: We obtain the inference time in the same way as the training time. We provide average inference time in Table VI (unit: millisecond). As shown in the table, GBDT-MO and GBDT-sparse are significantly faster than the others. The speed gap widens as the output dimension increases. The results are consistent with the inference complexity analysis in Section III-F. Note that we obtain the inference time by predicting the whole test set instead of an individual sample.

4) Memory Usage: GBDT-MO requires more memory than GBDT-SO because GBDT-MO needs to maintain the histograms of all output columns. Fortunately, the memory for them is much less than the memory for data sets in practice. Besides, memory usage highly depends on algorithm details and optimization. We compare memory usage of XGBoost, LightGBM, GBDT-sparse (our own implementation), GBDT-SO, and GBDT-MO in Table VII. It can be observed that GBDT-MO is better than XGBoost, LightGBM, and GBDT-sparse, while GBDT-SO is the best in terms of memory usage.

C. Sparse Split Finding

All experiments of GBDT-MO performed in Sections V-A and V-B use nonsparse split finding algorithm. In this section, we evaluate our unrestricted sparse split finding algorithm (see Algortithm 3) and restricted sparse split finding algorithm (see Algorithm 4). We compare their performance and training speed on MNIST, MNIST-inpainting, Caltech101, and NUS-WIDE with different sparse factor k. The hyperparameters used here are the same as the corresponding nonsparse one's.

Their training speed in seconds is shown in Table VIII. The restricted one is faster than the unrestricted one. Their speed differences are remarkable when k is large. The accuracy of the test samples is shown in Table IX. The restricted one is slightly better than the unrestricted one. Note that the accuracy of our sparse split algorithms is sometimes better than the nonsparse one's with a proper k (for example, $k = 64$ on Caltech101).

TABLE X
IMPACT OF DIAGONAL HESSIAN

Accuracy/Time	MNIST	Yeast
diagonal hessian	98.30/0.79s	62.29/0.0011s
exact hessian	98.17/3.96s	62.89/0.0094s

D. Objective With Exact Hessian

GBDT-MO replaces the exact hessian with the diagonal hessian to reduce computational complexity. We analyze the impact of the diagonal approximation on MNIST and Yeast because their output dimensions are relatively small. Table X shows accuracy and training time for a single round. Two methods have similar accuracy, but the diagonal approximation is much faster. Thus, the diagonal approximation of hessian achieves a good tradeoff between accuracy and speed.

VI. CONCLUSION

In this article, we have proposed a general method to learn GBDT for multiple outputs. The motivation of GBDT-MO is to capture the correlations between output variables and reduce the redundancy of tree structures. We have derived the approximated learning objective for both nonsparse case and sparse case based on the second-order Taylor expansion of loss. For the sparse case, we have proposed the restricted algorithm that restricts the subsets of left and right parts to be the same and the unrestricted algorithm that has no such a restriction. We have extended the histogram approximation into the multiple-output case to speed up training. We have evaluated that GBDT-MO is remarkably and consistently better in generalization ability and faster in both training and inference compared with GBDT for a single output. We have also evaluated that the restricted sparse split finding algorithm is slightly better than the unrestricted one by considering both performance

TABLE XI
GRID SEARCHED LEARNING RATE AND MAXIMUM DEPTH

Learning rate/Maximum depth	MNIST	Yeast	Caltech101	NUS-WIDE	MNIST-inpaining	Student-por
XGBoost	0.10/8	0.10/4	0.10/8	0.10/8	0.10/8	0.10/4
LightGBM	0.25/6	0.10/4	0.10/9	0.05/8	0.10/8	0.05/4
GBDT-sparse	0.05/8	0.10/5	0.10/9	0.10/8	-	-
GBDT-SO	0.10/6	0.10/4	0.05/8	0.05/8	0.10/8	0.05/4
GBDT-MO	0.10/8	0.10/5	0.10/10	0.10/8	0.10/7	0.10/4

TABLE XII
PRELIMINARILY SEARCHED HYPERPARAMETERS

	MNIST	Yeast	Caltech101	NUS-WIDE	MNIST-inpaining	Student-por
Gain threshold	1e-3	1e-3	1e-3	1e-6	1e-6	1e-6
Min samples in leaf	16	16	16	4	4	4
Max bins	8	32	32	64	16	8

TABLE XIII
NUMBER OF TREES OF LEARNED MODELS

	MNIST	Yeast	Caltech101	NUS-WIDE	MNIST-inpaining	Student-por
XGBoost	1700	299.0	2234.0	5921	4880	133.4
LightGBM	1760	334.0	2692.0	11636	5618	219.6
GBDT-sparse	645	21.4	237.4	494	-	-
GBDT-SO	2180	183.0	3697.0	12051	6373	373.7
GBDT-MO	615	85.1	517.7	3546	2576	125.0

and training speed. However, GBDT-MO suffers from slower convergence, especially at the beginning of training. In our future work, we would improve its convergence speed.

APPENDIX A
APPROXIMATED OBJECTIVE

We suppose there exists $\gamma > 0$ such that

$$\gamma \mathbf{H}_{ii} \geq \sum_j |\mathbf{H}_{ij}| \quad \forall i. \tag{26}$$

That is, $\mathbf{H}$ is dominated by its diagonal elements. Then, we have

$$\begin{aligned} \mathbf{w}^T \mathbf{H} \mathbf{w} &= \sum_i \sum_j \mathbf{H}_{ij} \mathbf{w}_i \mathbf{w}_j \\ &\leq \frac{1}{2} \sum_i \sum_j |\mathbf{H}_{ij}| \left(\mathbf{w}_i^2 + \mathbf{w}_j^2\right) \\ &= \sum_i \sum_j |\mathbf{H}_{ij}| \mathbf{w}_i^2 \\ &\leq \gamma \sum_i \mathbf{H}_{ii} \mathbf{w}_i^2. \end{aligned} \tag{27}$$

The last inequality holds based on the assumption in (26). Substituting this result into the Taylor expansion of loss

$$\begin{aligned} l(\hat{\mathbf{y}} + \mathbf{w}, \mathbf{y}) &\approx l(\hat{\mathbf{y}}, \mathbf{y}) + \mathbf{g}^T \mathbf{w} + \frac{1}{2} \mathbf{w}^T \mathbf{H} \mathbf{w} \\ &\leq l(\hat{\mathbf{y}}, \mathbf{y}) + \sum_i \mathbf{g}_i \mathbf{w}_i + \frac{\gamma}{2} \sum_i \mathbf{H}_{ii} \mathbf{w}_i^2. \end{aligned} \tag{28}$$

We ignore the remainder term of the Taylor expansion [9]. Substituting this upper bound of $l(\hat{\mathbf{y}} + \mathbf{w}, \mathbf{y})$ into (12), we get the optimal value of $\mathbf{w}$

$$\mathbf{w}_j^* = -\frac{\sum_i (\mathbf{g}_j)_i}{\gamma \sum_i (\mathbf{h}_j)_i + \lambda}. \tag{29}$$

Recall that $\mathbf{h}$ is the diagonal elements of $\mathbf{H}$. This solution is the same as (15), except the coefficient γ. In practice, the actual leaf weight is $\alpha \mathbf{w}_j^*$, where α is the so-called learning rate. Then, the effect of γ can be canceled out by adjusting α and λ. Thus, we do not need to consider the exact value of γ in practice.

APPENDIX B
HYPERPARAMETER SETTINGS

We discuss our hyperparameter settings. We find the best maximum depth d and learning rate via a grid search. d is searched from $\{4, 5, 6, 7, 8, 9, 10\}$, and learning rate is searched from $\{0.05, 0.1, 0.25, 0.5\}$. We provide the selected d and learning rate in Table XI. We set L_2 regularization λ to 1.0, and the maximum leaves to 0.75×2^d in all experiments. We preliminarily search other hyperparameters and fix them for all methods. We list them on real data sets in Table XII. Note that the hyperparameters in Table XII may not be optimal. When comparing with the gain threshold, we use the average gain over the involved output variables. We provide the number of trees of the best models in Table XIII. For single-variable GBDT models, the number of trees is the summation of the number of trees for each output variable.

APPENDIX C
STATISTICAL TEST OF RESULTS

There are no standard training and testing splits on Yeast, Caltech101, and Student-por. We obtain the results in Table V by averaging the results of ten random trials. Thus, statistical tests on how likely GBDT-MO is better than others are necessary. Denote X is a random variable that means the performance differences between GBDT-MO and the others. X is commonly supposed to be Student's T-distribution because its

TABLE XIV
CONFIDENCE OF THE SUPERIORITY OF GBDT-MO

Confidence	Yeast	Caltech101	Student-por
XGBoost	0.067	0.930	0.888
LightGBM	0.745	0.988	0.685
GBDT-SO	0.812	0.955	0.561

true variance is unknown. We estimate the mean and standard deviation of X from the ten random trials. Then, the confidence in whether GBDT-MO performs better than the others is defined as $P(X > 0)$ for classifications and $P(X < 0)$ for regressions, as reported in Table XIV. Most confidence scores are significantly higher than 0.5. Thus, it can be expected that GBDT-MO performs better than others.

REFERENCES

[1] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *Ann. Statist.*, vol. 29, no. 5, pp. 1189–1232, 2001.

[2] J. H. Friedman, "Stochastic gradient boosting," *Comput. Statist. Data Anal.*, vol. 38, no. 4, pp. 367–378, Feb. 2002.

[3] P. Li, "Robust logitboost and adaptive base class (ABC) logitboost," in *Proc. 26th Conf. Uncertainty Artif. Intell.*, 2010, pp. 302–311.

[4] C. Qi, A. Fourie, Q. Chen, X. Tang, Q. Zhang, and R. Gao, "Data-driven modelling of the flocculation process on mineral processing tailings treatment," *J. Cleaner Prod.*, vol. 196, pp. 505–516, Sep. 2018.

[5] C. J. Burges, "From ranknet to lambdarank to lambdamart: An overview," *Learning*, vol. 11, nos. 23–581, p. 81, 2010.

[6] S. Jiang, H. Mao, Z. Ding, and Y. Fu, "Deep decision tree transfer boosting," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 2, pp. 383–395, Feb. 2020.

[7] M. Richardson, E. Dominowska, and R. J. Ragno, "Predicting clicks: Estimating the click-through rate for new ads," in *Proc. Int. Conf. World Wide Web*, 2007, pp. 521–530.

[8] J. Bennett *et al.*, "The netflix prize," in *Proc. KDD Cup Workshop*, New York, NY, USA, 2007, p. 35.

[9] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2016, pp. 785–794.

[10] G. Ke *et al.*, "LightGBM: A highly efficient gradient boosting decision tree," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 3149–3157.

[11] L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin, "CatBoost: Unbiased boosting with categorical features," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 6638–6648.

[12] G. Tsoumakas, E. Spyromitros-Xioufis, J. Vilcek, and I. Vlahavas, "Mulan: A java library for multi-label learning," *J. Mach. Learn. Res.*, vol. 12, pp. 2411–2414, Jun. 2011.

[13] H. Borchani, G. Varando, C. Bielza, and P. Larrañaga, "A survey on multi-output regression," *Wiley Interdiscipl. Rev., Data Mining Knowl. Discovery*, vol. 5, no. 5, pp. 216–233, Sep. 2015.

[14] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org

[15] G. E. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," 2015, *arXiv:1503.02531*. [Online]. Available: https://arxiv.org/abs/1503.02531

[16] P. Geurts, L. Wehenkel, and F. d'Alché-Buc, "Gradient boosting for kernelized output spaces," in *Proc. 24th Int. Conf. Mach. Learn. (ICML)*, 2007, pp. 289–296.

[17] S. Si, H. Zhang, S. S. Keerthi, D. Mahajan, I. S. Dhillon, and C. Hsieh, "Gradient boosted decision trees for high dimensional sparse output," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 3182–3190.

[18] S. Tyree, K. Q. Weinberger, K. Agrawal, and J. Paykin, "Parallel boosted regression trees for Web search ranking," in *Proc. 20th Int. Conf. World Wide Web (WWW)*, 2011, pp. 387–396.

[19] T. Chen, S. Singh, B. Taskar, and C. Guestrin, "Efficient second-order gradient boosting for conditional random fields," in *Proc. Int. Conf. Artif. Intell. Statist.*, 2015, pp. 147–155.

[20] R. Agrawal, A. Gupta, Y. Prabhu, and M. Varma, "Multi-label learning with millions of labels: Recommending advertiser bid phrases for Web pages," in *Proc. 22nd Int. Conf. World Wide Web*, 2013, pp. 13–24.

[21] Y. Prabhu and M. Varma, "FastXML: A fast, accurate and stable tree-classifier for extreme multi-label learning," in *Proc. 20th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2014, pp. 263–272.

[22] R. E. Schapire and Y. Singer, "Improved boosting algorithms using confidence-rated predictions," in *Proc. 11th Annu. Conf. Comput. Learn. Theory (COLT)*, vol. 37, 1998, pp. 80–91.

[23] Y. Amit, O. Dekel, and Y. Singer, "A boosting algorithm for label covering in multilabel problems," in *Proc. Int. Conf. Artif. Intell. Statist.*, 2007, pp. 27–34.

[24] J. H. Friedman, "Multivariate adaptive regression splines," *Annu. Statist.*, vol. 19, no. 1, pp. 1–67, Mar. 1991.

[25] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, vol. 1, Jun. 2005, pp. 886–893.

[26] T.-S. Chua, J. Tang, R. Hong, H. Li, Z. Luo, and Y.-T. Zheng, "NUS-WIDE: A real-world Web image database from National University of Singapore," in *Proc. ACM Int. Conf. Image Video Retr.*, Santorini, Greece, Jul. 2009, pp. 1–9.

[27] E. Spyromitros-Xioufis, S. Papadopoulos, I. Y. Kompatsiaris, G. Tsoumakas, and I. Vlahavas, "A comprehensive study over VLAD and product quantization in large-scale image retrieval," *IEEE Trans. Multimedia*, vol. 16, no. 6, pp. 1713–1728, Oct. 2014.

Zhendong Zhang (Graduate Student Member, IEEE) received the B.S. degree in materials engineering from Dalian Jiaotong University, Dalian, China, in 2014. He is currently pursuing the Ph.D. degree in electronic engineering with Xidian University, Xi'an, China.

His main research interests include data mining and machine learning (including deep learning) and its applications to computer vision tasks.

Cheolkon Jung (Member, IEEE) received the B.S., M.S., and Ph.D. degrees in electronic engineering from Sungkyunkwan University, Seoul, South Korea, in 1995, 1997, and 2002, respectively.

He was with the Samsung Advanced Institute of Technology (Samsung Electronics), Suwon, South Korea, as a Research Staff Member, from 2002 to 2007. He was a Research Professor with the School of Information and Communication Engineering, Sungkyunkwan University, from 2007 to 2009. Since 2009, he has been with the School of Electronic Engineering, Xidian University, Xi'an, China, where he is currently a Full Professor and the Director of the Xidian Media Laboratory. His main research interests include image and video processing, computer vision, pattern recognition, machine learning, computational photography, video coding, virtual reality, information fusion, multimedia content analysis and management, and 3DTV.