DeltaBoost: Gradient Boosting Decision Trees with Efficient Machine Unlearning

ZHAOMIN WU, National University of Singapore, Singapore
JUNHUI ZHU, National University of Singapore, Singapore
QINBIN LI, National University of Singapore, Singapore
BINGSHENG HE, National University of Singapore, Singapore

As machine learning (ML) has been widely developed in real-world applications, the privacy of ML models draws an increasing concern. In this paper, we study how to forget specific data records from ML models to preserve the privacy of these data. Although some studies propose efficient unlearning algorithms on random forests and extremely randomized trees, Gradient Boosting Decision Trees (GBDT), which are widely used in practice, have not been explored. The efficient unlearning of GBDT faces two major challenges: 1) the training of each tree is deterministic and non-robust; 2) the training of a tree depends on all the previous trees. To solve the first challenge, we propose a robust GBDT-like ML model *DeltaBoost* that enables efficient and accurate deletion according to our theoretical analysis. For the second challenge, we design a training algorithm for DeltaBoost that minimizes the dependency among trees. Our experiments on five datasets demonstrate that DeltaBoost can remove data records from the trained model efficiently and effectively. Our unlearning approach achieves up to two orders of magnitude speedup compared to retraining GBDT. Besides, DeltaBoost produces competitive performance to existing decision-tree-based ML models.

CCS Concepts: • **Computing methodologies** → **Boosting**; **Classification and regression trees**; **Bagging**; • **Security and privacy** → **Pseudonymity, anonymity and untraceability**; *Privacy-preserving protocols.*

Additional Key Words and Phrases: gradient boosting decision trees; machine unlearning; data deletion

ACM Reference Format:
Zhaomin Wu, Junhui Zhu, Qinbin Li, and Bingsheng He. 2023. DeltaBoost: Gradient Boosting Decision Trees with Efficient Machine Unlearning. *Proc. ACM Manag. Data* 1, 2, Article 168 (June 2023), 26 pages. https://doi.org/10.1145/3589313

1 INTRODUCTION

Recent data management applications usually involve machine learning models trained from the data. These models, though providing convenient access to the knowledge of the data, introduce additional privacy concerns. To address these concerns, new laws and regulations have been made recently in many countries or regions, such as General Data Protection Regulation (GDPR) [25] and California Consumer Privacy Act (CCPA) [15]. These laws demand that users can request a company to delete their personal information (a.k.a., "right to be forgotten"). In real applications, personal information is stored not only in databases but also in machine learning models. *Machine unlearning* [8, 12, 27], studying how to make machine learning models forget specific sensitive data, has become an emerging research topic recently.

Authors' addresses: Zhaomin Wu, zhaomin@comp.nus.edu.sg, National University of Singapore, Singapore; Junhui Zhu, junhui.zhu@comp.nus.edu.sg, National University of Singapore, Singapore; Qinbin Li, liqinbin1998@gmail.com, National University of Singapore, Singapore; Bingsheng He, hebs@comp.nus.edu.sg, National University of Singapore, Singapore.

2836-6573/2023/6-ART168
https://doi.org/10.1145/3589313

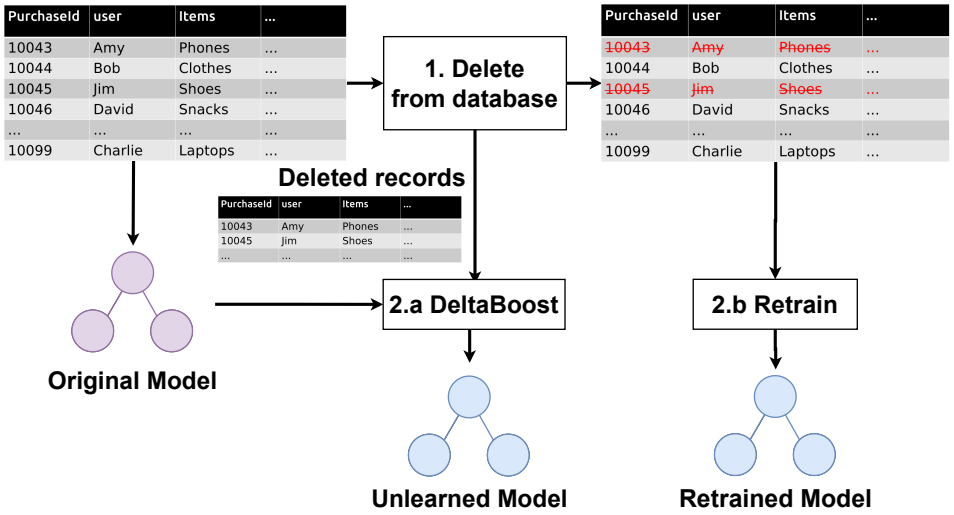


Fig. 1. Illustration of Machine Unlearning

The procedure of machine unlearning can be explained by an example in Fig. 1. Imagine a shopping website with a backend database that stores users' shopping records. The recommendation system of the website uses these records to train a machine learning model to recommend products as users browse the website. When some users request the website to delete their accounts, the website's backend needs to delete all the shopping records related to the users to ensure information reciprocity, thus generating a delete request on the database. Besides conducting deletion on the database, the model trained on the original data also contains information about the users. A trivial approach is to delete the original model and retrain a new machine learning model on the remained dataset with specific records removed (e.g., 2.b in Fig. 1). Nonetheless, retraining is very time-consuming, which could be unacceptable, especially when providing real-time services to the user. Therefore, in many applications, efficient machine unlearning algorithms are desired to handle deletion requests in low latency (e.g., 2.a in Fig 1).

Gradient boosting decision trees (GBDT) [10] is a popular machine learning model that features efficiency and explainability. Nevertheless, to the best of our knowledge, no unlearning algorithms have been developed for GBDT. A general approach [8] supports decision trees, but it suffers an efficiency issue since the unlearning algorithm is unaware of the tree structure. Some studies investigate the unlearning algorithms on random forests [7] and extremely randomized trees [27]. Both approaches rely heavily on the random choice of split candidates, which does not apply to GBDT, which deterministically selects the split value with the maximum gain for better performance. Moreover, in both approaches, the training and deletion of each tree are independent, which no longer holds for GBDT, where a tree is affected by all the previous trees. Meanwhile, supposed α is the number of deletion requests, we observe that GBDT requires about 107α seconds for retraining on a dataset (`msd` in Section 6.1) with 500k instances and 90 features, which could be unacceptable in practice. Hence, an unlearning approach is desired to efficiently delete data records from GBDT.

We find that GBDT has two challenges in data removal due to its deterministic and boosting properties. First, in GBDT, removing a few instances will change the gain of each possible split candidate, thus affecting the selected best split value of the current internal node and the training of the subtree starting from the internal node. Second, unlike random forests, the training of a

tree depends on all the previous trees in GBDT. When a leaf weight of a tree changes due to data deletion, the gradients of the corresponding instances changes and all the later trees will be affected. To enable efficient unlearning without updating the entire model from scratch, a robust GBDT algorithm is in demand.

To address the first challenge, we propose *bin-tree* to construct a robust histogram, which ensures that deleting instances does not generate new split candidates. Then, we theoretically analyze how removing random instances affects the choice of split points. Our analysis suggests that when only a small fraction of instances are removed, the split point with the maximum gain shifts in a small neighborhood. Based on this observation, we store a *split neighborhood* containing neighboring split candidates instead of a single split point in each node. After deletion, we re-calculate gains and re-select the best split candidate in the neighborhood.

We minimize the negative impact by random bagging and quantization to address the second challenge. 1) Bagging [11] was initially designed to prevent over-fitting and speed up training by training each tree by a sampled batch instead of the global training set in GBDT. In this study, we find bagging can also help dilute the effect of changed leaf weights in the later trees. With random sampling, each instance in a changed leaf node only affects a portion of trees. 2) Gradient quantization [28] was initially proposed for the model compression of GBDT. In [28], the authors demonstrate that the quantization of gradient and Hessian into two to four bits hardly affects the model performance. In our study, we observe that a small change in the original leaf weight hardly affects the quantized gradient and Hessian. Thus, gradient quantization can be adopted to reduce the dependency among trees.

The source codes of DeltaBoost have been released in a repository[1]. Our main contributions are summarized as follows.

- We theoretically analyze the effect on the best split value by randomly removing instances, which motivates our robust GBDT design.
- We propose a novel GBDT-like algorithm named *DeltaBoost*, with new training techniques including robust training algorithm and efficient data removal algorithms.
- We conduct experiments on five datasets to demonstrate the efficacy and efficiency of DeltaBoost. The results indicate that DeltaBoost enables efficient and effective data unlearning (at most 446.8× speedup compared to retraining GBDT) while producing a close performance to existing tree-based ML algorithms.

The following part of this paper is organized as follows. Section 2 introduces the GBDT model along with some mathematical theorems. Section 3 describes the reason why GBDT is difficult to meet for efficient unlearning and corresponding theoretical analysis. Section 4 introduces the structure of DeltaBoost. Section 5 discusses the learning and unlearning algorithms of DeltaBoost, as well a proposed metric of unlearning. Section 6 shows the results of experiments, section 7 summarizes the current works related to machine unlearning, and section 8 concludes the work of this paper. The detailed proofs are included in a technical report [32].

2 PROBLEM FORMULATION AND BACKGROUND

2.1 Problem Formulation

Consider a dataset $\mathbf{D} = \{\mathbf{x}_i, y_i\}_{i=1}^{n} \in \mathcal{D}$ contains n instances and labels. Each instance $\mathbf{x}_i = \left[x_i^{(1)}, ..., x_i^{(m)}\right]$ contains m features. Suppose a subset $\mathbf{D}_d$ of $\mathbf{D}$ is requested to be deleted from $\mathbf{D}$. The corresponding remained dataset is denoted as $\mathbf{D}_r = \mathbf{D} \setminus \mathbf{D}_d$. We aim to design a machine learning model $M \in \mathcal{M}$ based on GBDT and its corresponding learning/unlearning algorithms

[1]https://github.com/Xtra-Computing/DeltaBoost

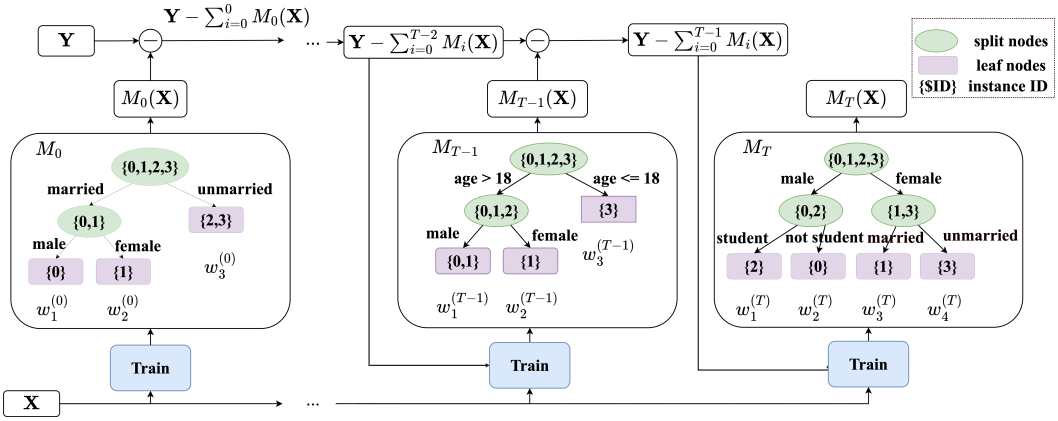


Fig. 2. Training process of GBDT

f_{learn}, $f_{unlearn}$. Specifically, $f_{learn} : \mathcal{D} \rightarrow \mathcal{M}$ learns a machine learning model from a dataset. $f_{unlearn} : \mathcal{M} \times \mathcal{D} \times \mathcal{D} \rightarrow \mathcal{M}$ deletes data from a model to obtain a new model. For simplicity, we define $M = f_{learn}(\mathbf{D})$ as the *original model*, $M_r = f_{learn}(\mathbf{D}_r)$ as the remained model (i.e., the model trained on the remained dataset), $M_d = f_{unlearn}(M, \mathbf{D}, \mathbf{D}_d)$ as the deleted model (i.e., the model updated from original model through model unlearning on the deleted dataset). Our goal is to design an efficient approach for data deletion while M_r and M_d are approximately the same, i.e.,

$$f_{unlearn}(f_{learn}(\mathbf{D}), \mathbf{D}, \mathbf{D}_d) = M_d \approx f_{learn}(\mathbf{D} \setminus \mathbf{D}_d) = M_r$$

Deletion algorithms that ensure $M_d = M_r$ are called *exact deletion* [12], which is usually more expensive and not always necessary. This study, like other approaches [16, 27, 31], focuses on *approximate deletion* formally defined in [16]. This ensures that the probabilistic distribution of M_d and M_r are similar, i.e., $e^{-\varepsilon} \leq P(M_d)/P(M_r) \leq e^{\varepsilon}$, where ε is a small positive constant value.

2.2 Gradient Boosting Decision Trees

Gradient boosting decision trees (GBDT) [10] is a tree-based machine learning model that minimizes the objective function by gradient descent based on the residual between the prediction value and the label. The training process of GBDT is summarized in Fig. 2. In the first iteration, a decision tree is trained to predict the real labels $\mathbf{y}$. In the second iteration, a decision tree is trained to fit the residual $(\mathbf{y} - \hat{\mathbf{y}}_1)$ of the previous model, where $\hat{\mathbf{y}}_1$ is the prediction value of the first tree. Formally, the t-th tree of the GBDT model denoted as M_t is trained by minimizing the objective function defined as Equation 1.

$$L^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}^{(t-1)} + M_t(\mathbf{x}_i)) + \Omega(M_t), \tag{1}$$

where $\Omega(M_t)$ is the regularization term, $\hat{y}^{(t-1)}$ is the prediction of $t-1$ previous trees, l is the loss function (e.g., mean square error).

Each tree minimizes the loss by recursively splitting the nodes starting from the root. For each node to be split, GBDT proposes $\sum_{i=1}^{m} c_i$ split candidates $\left\{s_1^{(1)}, ..., s_{c_1}^{(1)}, s_1^{(2)}, ..., s_{c_m}^{(m)}\right\}$ (m features and c_i splitting candidates for every feature) that split the instances in the node to two child nodes. Taking a split candidate $s_i^{(a)}$ on feature a as an example, define $I_{Left} = \{x_i | x_i^{(a)} < s_i^{(a)}\}$

and $I_{Right} = \{x_i | x_i^{(a)} \geq s_i^{(a)}\}$ as the instance sets of the left and right child, respectively. Let $I = I_{Left} \cup I_{Right}$ be the instance set in the node. The gain of $s_i^{(a)}$ is calculated according to Equation 2.

$$\phi_0(s_i^{(a)}) = \frac{1}{2}\left[\frac{(\sum_{i\in I_{Left}} g_i)^2}{\sum_{i\in I_{Left}} h_i + \lambda} + \frac{(\sum_{i\in I_{Right}} g_i)^2}{\sum_{i\in I_{Right}} h_i + \lambda} - \frac{(\sum_{i\in I} g_i)^2}{\sum_{i\in I} h_i + \lambda}\right] - \gamma \tag{2}$$

where $g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$ and $h_i = \partial^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$ are the gradient and Hessian statistics on the loss function, and λ, γ are hyperparameters for regularization. The split candidate with the maximum gain is selected as the split point of this node. Finally, a node stops splitting if the best gain is negative or reaches the given maximum depth of the tree. The predicted value of all the instances on leaf node I is calculated by Equation 3.

$$w_I = -\frac{\sum_{i\in I} g_i}{\sum_{i\in I} h_i + \lambda} \tag{3}$$

Among many previous studies [10, 17, 21, 30] to find split candidates in each node, the histogram-based method is widely used. Since the split points found by histogram methods are less affected after deletion, we focus on histogram methods in DeltaBoost.

The overall training algorithm is shown in Algorithm 1. We first update the gradients and Hessians of the instances and build a tree with the gradients (lines 14-16). When building a tree, the gain of each possible split candidate is computed by Equation 2, and the split candidate with the maximum gain is selected as the split point to split the current instance set (lines 5-7). When reaching the maximum depth or the node weight is too small, the current node is set to a leaf node, and the leaf value is computed by Equation 3 (lines 2-4). The prediction value is updated after building the tree (lines 15-16).

In the inference, the final prediction of $\mathbf{x}$ is the summation of the prediction of all trees (assuming the learning rate is 1) as Equation 4.

$$M(\mathbf{x}) = \sum_{t=1}^{T} M_t(\mathbf{x}) \tag{4}$$

2.3 Probabilistic Foundations

This subsection introduces some useful mathematical tools to support our analysis. Since the analysis is based on some independence assumptions as elaborated in Section 3, we require some theorems to derive targeting expected values. The expected value and variance of the sum of independent random variables can be easily calculated by Theorem 2.1 and Theorem 2.2.

Theorem 2.1. *([24]) Let $X_1, X_2, ..., X_n$ be independent random variables. Then, $Var\left(\sum_{i=1}^n X_i\right) = \sum_{i=1}^n Var(X_i)$.*

Theorem 2.2. *([24]) Let $X_1, X_2, ..., X_n$ be independent random variables. Then, $\mathbb{E}\left[\sum_{i=1}^n X_i\right] = \sum_{i=1}^n \mathbb{E}[X_i]$.*

To further analyze the expected gain value, we also need an approximation of the ratio of two random variables.

Lemma 2.3. *([5]) For random variable X and Y, if $Y \in [0, \infty)$, we have the following approximation*

$$\mathbb{E}\left[\frac{X}{Y}\right] = \frac{\mathbb{E}[X]}{\mathbb{E}[Y]} + O\left(\frac{Var(Y)\mathbb{E}[X]}{\mathbb{E}[Y]^3} - \frac{Cov(X,Y)}{\mathbb{E}[Y]^2}\right)$$

Algorithm 1: Training of GBDT

Data: Data $\mathbf{x}$, labels $\mathbf{y}$, min node weights h_{min}, max depth κ of trees
function *TrainTree*($\mathbf{x}$,$\mathbf{g}$,$\mathbf{h}$)**:**
  if $\sum_{h_i \in \mathbf{h}} h_i < h_{min}$ ***or*** *current depth* $> \kappa$ **then**
    return Leaf with weight w_T calculated by Equation 3
  end
  $\Phi_0 \leftarrow$ calculate gain $\phi_0(s_i^{(a)})$ for each possible split candidate $s_i^{(a)}$ by Equation 2;
  $s^* \leftarrow$ split point with max gain in all split candidates S;
  $(\mathbf{x}_l, \mathbf{g}_l, \mathbf{h}_l), (\mathbf{x}_r, \mathbf{g}_r, \mathbf{h}_r) \leftarrow$ split $\mathbf{x}, \mathbf{g}, \mathbf{h}$ at s^*;
  $T_l \leftarrow \textsc{TrainTree}(\mathbf{x}_l, \mathbf{g}_l, \mathbf{h}_l)$, $T_r \leftarrow \textsc{TrainTree}(\mathbf{x}_r, \mathbf{g}_r, \mathbf{h}_r)$;
  return node split at s^* and subtrees T_l, T_r ;
function *TrainGBDT*($\mathbf{x}$, $\mathbf{y}$)**:**
  $M_0 \leftarrow \emptyset$, $L^{(0)} \leftarrow$ Equation 1 with $\mathbf{y}$;
  for $t \leftarrow 1$ *to* t_{max} **do**
    $\mathbf{g}_t, \mathbf{h}_t \leftarrow$ calculated from $L^{(t)}$;
    $T_t \leftarrow \textsc{TrainTree}(\mathbf{x}_t, \mathbf{g}_t, \mathbf{h}_t)$;
    $M_t \leftarrow M_{t-1} \cup \{T_t\}$;
    $L^{(t)} \leftarrow$ update loss function with M_t and $\mathbf{y}$ (Equation 1);
  end
  return $M_{t_{max}}$;

Lemma 2.3 is approximated by the first-order Taylor series of the expected value. Our empirical results indicate that it is sufficiently accurate for estimating the expected gain value.

3 OBSERVATION AND MOTIVATION

In this section, we illustrate why GBDT is challenging for efficient machine unlearning, followed by a theoretical analysis that directs our amendment towards robust GBDT. To simplify the theoretical analysis, we have made the following three assumptions. Note that our proposed algorithm design does not rely on these assumptions, as we will discuss in Section 4.

- *Selecting the records to delete is approximated as a sampling with replacement.* This approximation hardly affects expected values as well as our conclusions if the proportion of deletions is small (typically < 10% suffices [18] in many applications). For a large amount of deletions (which is not the focus of this paper), retraining could be a more suitable solution.
- *The feature values are independent across records.* We regard these data as values instead of random variables like existing approaches [7, 27].
- *The deletion requests are independent.* Although there may be some correlated deletions in practice, we believe this assumption is still reasonable for applications with large amount of training data.

3.1 Observation

GBDT is challenging for unlearning. Considering a single decision tree, if the removed instances affect the selected split point of an internal node, the data distribution of its child nodes might be fundamentally changed, especially when the split point is selected on a new feature. Even if the new split point after removal is only changed by a small value on the same feature, such slight changes accumulate as the depth increases, resulting in significantly different leaf nodes. Considering the

boosting process, the change of each decision tree after removal also accumulates during the boosting. These challenges imply that GBDT is not robust to data removal, which motivates our design of a robust GBDT variant.

Removing a small fraction of instances hardly affects the choice of the split point for the best gain. When the number of removed instances $n_d \ll n$, we theoretically prove that, for most datasets, the split point with the maximum gain only shifts in its neighborhood under a small fraction of random deletion. Although there may be some correlated deletions in practice, we believe this assumption is still reasonable for applications with large amount of training data.

We formally summarize the process of random deletion as follows. For any feature a, suppose n_d feature values are randomly selected from $\{x_i^{(a)}\}_{i=1}^n$ for deletion. Suppose there are m split candidates $s_1^{(a)}, ..., s_{m+1}^{(a)}$. The original instances can be categorized into m index sets $B_1^{(a)}, ..., B_m^{(a)}$, which satisfy that $\forall i \in B_j^{(a)}, s_j^{(a)} \leq x_i^{(a)} < s_{j+1}^{(a)}$. Meanwhile, the deleted instances can also be categorized into m sets $d_1^{(a)}, ..., d_m^{(a)}$ which satisfy that $\forall i \in d_j^{(a)}, s_j^{(a)} \leq x_i^{(a)} < s_{j+1}^{(a)}$. It is worth mentioning that for a trained model, the distribution of B is determined. However, for a series of random sample deletions, the distribution of d is random. These index sets are called *bins*. These bins, as well as the sum of gradients in bins, are called a *histogram*. Since the analysis in this section is based on a single feature a, for simplicity, we denote $B_j^{(a)}$ and $d_j^{(a)}$ as B_j and d_j, respectively.

After the removal, the gain of split value s_k is calculated by

$$\begin{aligned}\phi_1(d_1, ..., d_m; s_k) = &\frac{\left(\sum_{j=1}^{k-1}\sum_{i\in B_j} g_i - \sum_{j=1}^{k-1}\sum_{i\in d_j} g_i\right)^2}{\sum_{j=1}^{k-1}\sum_{i\in B_j} h_i - \sum_{j=1}^{k-1}\sum_{i\in d_j} h_i + \lambda} + \\ &\frac{\left(\sum_{j=k}^{m}\sum_{i\in B_j} g_i - \sum_{j=k}^{m}\sum_{i\in d_j} g_i\right)^2}{\sum_{j=k}^{m}\sum_{i\in B_j} h_i - \sum_{j=k}^{m}\sum_{i\in d_j} h_i + \lambda} - \\ &\frac{\left(G - \sum_{j=1}^{m}\sum_{i\in d_j} g_i\right)^2}{H - \sum_{j=k}^{m}\sum_{i\in d_j} h_i + \lambda},\end{aligned} \tag{5}$$

where $G = \sum_{j=1}^{m}\sum_{i\in B_j} g_i$ and $H = \sum_{j=1}^{m}\sum_{i\in B_j} h_i$. With this definition, our finding can be formulated as Theorem 3.1. The assumptions in Theorem 3.1 are widely held in the empirical studies in Section 3.3.

Theorem 3.1. *Suppose* $\lambda \ll \sum_{j=1}^{k-1}\sum_{i\in B_j} h_i$, $\lambda \ll \sum_{j=k-1}^{m}\sum_{i\in B_j} h_i$, *and* $n_d \ll n$. *For* $\forall s_k (k \in [1, m])$,

$$\arg\max_k \phi_1(d_1, ..., d_m; s_k) \approx \arg\max_k \phi_0(s_k)$$

if $\zeta n_d \ll n$, *where* $\zeta = \frac{a_r c_l^2 + a_l c_r^2}{(b_r c_l - b_l c_r)^2}$,

$$\begin{aligned} a_l &= \sum_{j=1}^{k-1}\sum_{i\in B_j} g_i^2,\ b_l = \sum_{j=1}^{k-1}\sum_{i\in B_j} g_i,\ c_l = \sum_{j=1}^{k-1}\sum_{i\in B_j} h_i \\ a_r &= \sum_{j=k}^{m}\sum_{i\in B_j} g_i^2,\ b_r = \sum_{j=k}^{m}\sum_{i\in B_j} g_i,\ c_r = \sum_{j=k}^{m}\sum_{i\in B_j} h_i \end{aligned}$$

In the following, we briefly present the theoretical analysis for Theorem 3.1. The detailed proof of Theorem 3.1, the approximation error analysis, and other supporting analyses are included in the technical report [32].

3.2 Theoretical Analysis

Overview. We gives an outline of the proof of Theorem 3.1 in this subsection. Since the gain $\phi_1(d_1, ..., d_m; s_k)$ is composed of (a) the sum of random variables, and (b) the sum of squared random variables. We first calculate the expected values of (a) and (b) by studying a sub-problem "sum of values in a single bin". Then, with these results, we estimate the expected value of the gain by first-order Taylor series (Lemma 2.3). Under the given assumptions in Theorem 3.1, the gradient of this expected value w.r.t. the remove ratio can be proved to be a constant that is proportional to the gain value. Therefore, we derive that the best split candidate with the max gain value is expected to remain unchanged. The notations used in the analysis is summarized in Table 1.

Table 1. Mathematical notations

Symbol	Meaning
n	number of instances
n_d	number of instances to be deleted
s_i	value of i^{th} split candidate
B_j	index set of original instances in the j^{th} bin
d_j	index set of deleted instances in the j^{th} bin
g_i, h_i	gradient and Hessian of i^{th} instance in GBDT
λ	coefficient of L2 parametrization in the regular term
v_i	feature value of i^{th} instance
$\phi_0(s_k)$	gain of split value before removal at split point s_k
$\phi_1(s_k)$	gain of split value after removal at split points$_k$

We first study the following problem, which is helpful for calculating the expected value of gain.

Sum of deleted values in a single bin: Given a list of values $\{v_i\}_{i\in B_j}$ in bin B_j, if we randomly choose n_d values from the list with replacement, we need to calculate the distribution of the sum of the chosen values that fall into the j-th bin, i.e., $\sum_{i\in d_j} v_i$. (Note that v_i may refer to either gradient g_i or Hessian h_i in GBDT.)

$\mathbb{E}\left[\sum_{i\in d_j} v_i\right]$ can be derived according to the linearity of expected values according to Lemma 3.2.

Lemma 3.2. *For* $\forall S \subseteq [1, m] \cup \mathbb{N}$, *we have*

$$\mathbb{E}\left[\sum_{j\in S}\sum_{i\in d_j} v_i\right] = \frac{n_d}{n}\sum_{j\in S}\sum_{i\in B_j} v_i$$

Besides the expected value, we can calculate the second raw moment of $\sum_{i\in d_j} v_i$, i.e., $\mathbb{E}\left[\left(\sum_{i\in d_j} v_i\right)^2\right]$, which is very helpful for estimating the expected value of $\phi_1(d_1, ..., d_m; s_k)$.

Lemma 3.3. $\forall S \subseteq [1, m] \cup \mathbf{N}$, $\mathbb{E}\left[\left(\sum_{j\in S}\sum_{i\in d_j} v_i\right)^2\right] =$
$\frac{n_d}{n}\sum_{j\in S}\sum_{i\in B_j} v_i^2 + \frac{n_d^2 - n_d}{n^2}\left(\sum_{j\in S}\sum_{i\in B_j} v_i\right)^2$

Note that v_i can be replaced by either g_i or h_i. With Lemma 3.2 and Lemma 3.3, we can estimate the expected value of gain according to Theorem 3.4.

Theorem 3.4. *The expected value of the gain after removal on split value s_k can be estimated by*

$$\begin{aligned}\mathbb{E}[\phi_1(d_1,...,d_j;s_k)] \approx & \frac{a_l n_d + \frac{(n-n_d)^2-n_d}{n} b_l^2}{(n-n_d)c_l + n\lambda} + \frac{a_r n_d + \frac{(n-n_d)^2-n_d}{n} b_r^2}{(n-n_d)c_r + n\lambda} \\ & - \frac{(a_l+a_r) n_d + \frac{(n-n_d)^2-n_d}{n} (b_l+b_r)^2}{(n-n_d)(c_l+c_r) + n\lambda},\end{aligned} \tag{6}$$

where

$$\begin{aligned} a_l &= \textstyle\sum_{j=1}^{k-1} \sum_{i\in B_j} g_i^2,\ b_l = \sum_{j=1}^{k-1} \sum_{i\in B_j} g_i,\ c_l = \sum_{j=1}^{k-1} \sum_{i\in B_j} h_i \\ a_r &= \textstyle\sum_{j=k}^{m} \sum_{i\in B_j} g_i^2,\ b_r = \sum_{j=k}^{m} \sum_{i\in B_j} g_i,\ c_r = \sum_{j=k}^{m} \sum_{i\in B_j} h_i \end{aligned}$$

The remainder of this approximation is formally stated and proved in the technical report [32]. With Theorem 3.4, we can further investigate how random removal affects the gain by estimating the expected value. Theorem 3.5 demonstrates that, when $n_d \ll n$, as n_d increases, the gain of each split candidate changes at an almost constant rate determined by histogram.

Theorem 3.5. *Suppose $\lambda \ll \sum_{j=1}^{k-1} \sum_{i\in B_j} h_i$, $\lambda \ll \sum_{j=k-1}^{m} \sum_{i\in B_j} h_i$, and $n_d \ll n$. For $\forall s_k (k \in [1,m])$,*

$$\frac{\partial \mathbb{E}[\phi_1(d_1,...,d_j;s_k)]}{\partial n_d} \approx \alpha(s_k, \{g_i\}_{i=1}^n, \{h_i\}_{i=1}^n).$$

where α is unrelated to n_d, which is determined by the histogram and s_k.

Then, our main observation (Theorem 3.1) can be naturally proved according to the constant gradient.

3.3 Empirical Validation

Expected Values of ϕ_1. To demonstrate the accuracy of Equation 6 in estimating the expected value of gain after removal, we perform deletion 1000 times and compute the mean of $\phi_1(d_1,...,d_j;s_k)$ to obtain a ground-truth expected value. This ground-truth expected value is compared with the expected value calculated by Equation 6. The results are presented in Fig. 3. We observe that the "calculated" curve is overlapped with the "approximated" curve, indicating that our estimation is almost identical to the ground truth in all the split candidates.

Values of $\zeta n_d/n$. An important assumption in Theorem 3.1 is that $\zeta n_d/n$ is sufficiently small. For the five datasets in our experiments (see Section 6.1 for details), supposed $n_d/n = 1\%$, we evaluate the value of $\zeta n_d/n$ for each split candidate in the histogram. The mean and standard deviation of $\zeta n_d/n$ on the root node's split feature is presented in Table 2. We observe that $\zeta n_d/n$ is very small (< 0.01) in most datasets. This result empirically illustrates that this assumption widely holds in real-world data.

Values of λ. λ is a regularization hyperparameter in GBDT which can usually be set to zero with negligible performance loss [28]. Since Theorem 3.1 requires a sufficiently small λ, we set $\lambda = 0$ in all the experiments.

Table 2. Values of $\zeta n_d/n$ in real datasets

Dataset	codrna	covtype	gisette	cadata	msd
$\zeta n_d/n$	0.0005	0.2850	0.0021	0.0011	0.2359

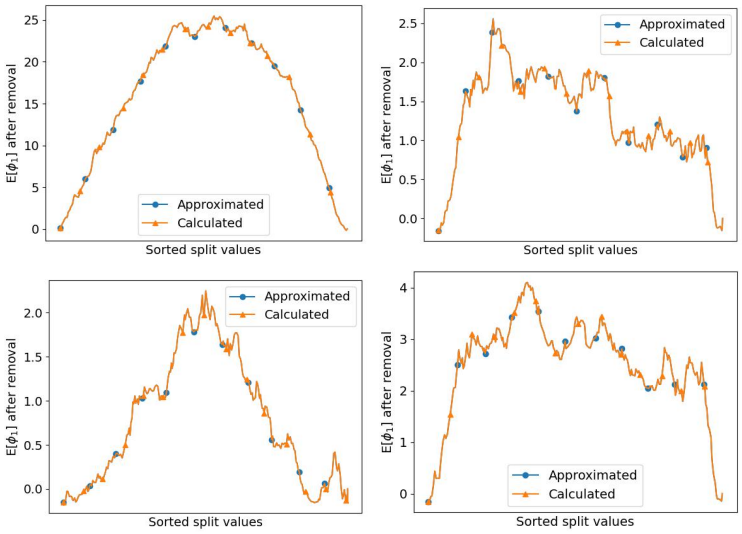


Fig. 3. Expected value of gain ϕ_1 on four features of the cadata dataset. "Calculated" means estimated by Equation 6; "Approximated" means approximated by the average of 1000 times deletion

Approximation Error. The error of this estimation is hard to be theoretically derived since the approximation involves `argmin` and derivative. Nonetheless, we can empirically validate this conclusion by comparing the curve of ϕ_0 and $\phi_0 - \phi_1$, the result of which is displayed in Fig. 4. From the figure, we clearly observe that the shifted gain (delta-gain in the figure) is almost proportional to the original gain. Therefore, the max value of gain remains almost unchanged in both cases, which empirically demonstrates the correctness of our conclusion.

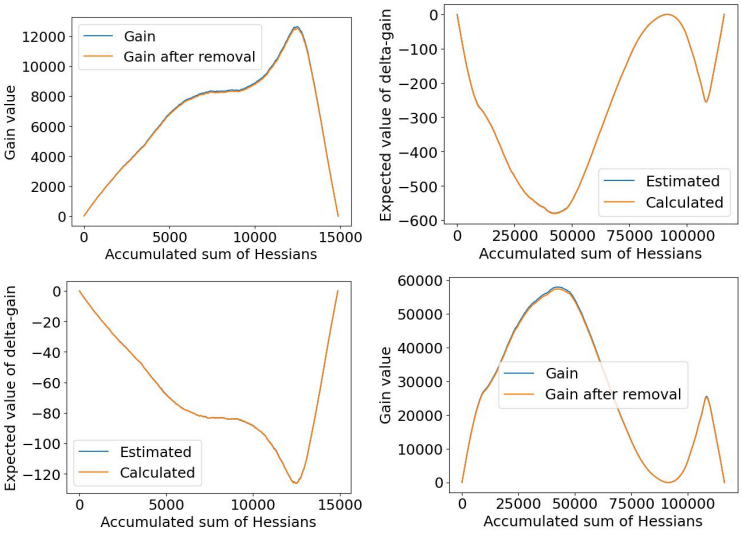


Fig. 4. Validation of Theorem 3.1 on cadata (column 1) and covtype (column 2). The first row is ϕ_0 and ϕ_1; the second row is $\phi_0 - \phi_1$.

4 MODEL STRUCTURE OF DELTABOOST

In this section, we introduce the structure of DeltaBoost. First, an overview of DeltaBoost is provided. Then, we introduce the main data structures and techniques in DeltaBoost, including balanced robust histogram (Section 4.1), split neighborhood (Section 4.2), robust feature selection (Section 4.3), and robust boosting (Section 4.4). Although we make three assumptions in theoretical analysis, our algorithm design does not have those assumptions. For example, we assume independent deletions in theoretical analysis, but our algorithm can support batch deletions.

Overview. We design DeltaBoost following the steps of training in GBDT. First, we construct a robust and balanced histogram from the data, which requires only slight adjustments to generate the retrained histogram after the deletion. Second, based on our analysis in Section 3, we design a robust data structure, namely split neighborhood, to select the best split candidate from the histogram in each feature. The split neighborhood ensures deleting a small number of instances hardly affects the tree structure. Third, we propose a robust approach to select the split feature that is less affected by deletion and produces good model accuracy.

4.1 Balanced Robust Histogram

The vanilla GBDT constructs a histogram on evenly distributed split candidates, i.e., the number of instances between neighboring split candidates is the same. However, this construction process is non-robust because a single record removal can affect at most all the split candidates (when deleting the first record). The opposite extreme case is to create a histogram by evenly dividing the range of feature values without considering data distribution, denoted as *feature-based*. Nonetheless, the histogram constructed by this approach is very imbalanced, causing a severe performance loss. Hence, we propose a robust histogram with relatively balanced data instances in this subsection.

Bin-Tree. For the convenience of updating after removal, we designed a tree-structured robust histogram, named *Bin Tree* as displayed in Fig. 5a and Fig. 5b. We split the instances recursively, and each node refers to a bin that covers a range of feature values. Specifically, a node contains five parameters: 1) *lower*: lower bound of feature values; 2) *upper*: upper bound of feature values; 3) *left*: pointer to the left child; 4) *right*: pointer to the right child; 5) *#ins*: number of instances in the range. After a bin-tree is built, the bins represented by all the leaf nodes form a dense partition of $[x_{min}^{(a)}, x_{max}^{(a)})$.

We formally summarize bin-trees building in Algorithm 2. Repeatedly, we find the bin B with max *#ins* (line 13) and split B at the mean of the feature range (lines 10-11). Meanwhile, we need to carefully handle cases where many instances have the same feature value by not splitting any bin with intervals more narrow than ϵ (line 13). If some instances are removed, a bin-tree can be updated in parallel (Algorithm 3) with the deleted feature values.

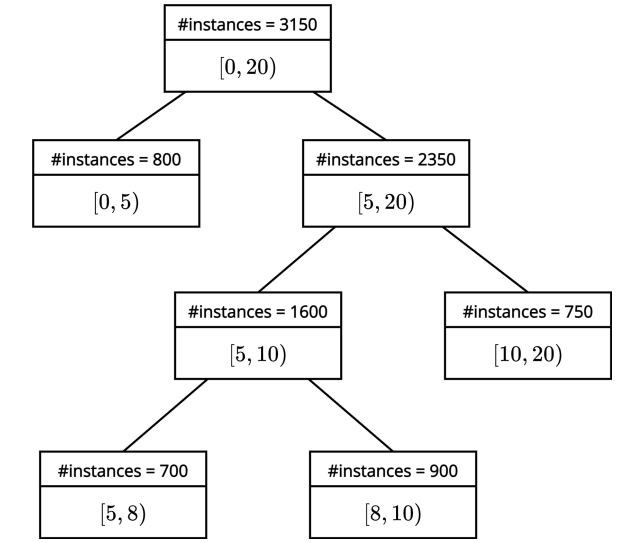
The bin-tree has three useful features. First, bin-tree guarantees that data deletion does not generate any new split candidates (Lemma 4.1). Moreover, after an efficient update in Algorithm 3, the updated bin-tree is exactly the same as the re-built bin-tree on the remained feature values (Theorem 4.2). Second, the update process (Algorithm 3) is efficient. Specifically, BuildHistogram takes $O(n\kappa)$ time and $O(n\kappa)$ space; RemoveFromTree (Algorithm 3) takes $O(n_d\kappa)$ time and $O(1)$ additional space, where κ is the depth of bin-tree. Third, as demonstrated in Fig. 6 generated from real data, DeltaBoost constructs a histogram containing bins with similar sizes to the bins of the GBDT histogram. In summary, the robust histogram ensures the consistency of split points, which makes a robust decision tree possible.

Lemma 4.1. *Suppose* $H = \textit{BuildHistogram}(\mathbf{x}^{(a)}, t)$ *and* $H_r = \textit{BuildHistogram}(\mathbf{x}^{(a)} \setminus \mathbf{x}_d^{(a)}, t)$. *If two nodes with the same lower and upper bounds are considered equal, it holds* $H_r \subseteq H$.

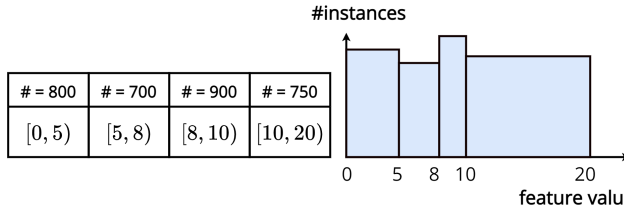
Proof. We prove this lemma by induction. 1) For a fixed feature range $(x_{min}^{(a)}, x_{max}^{(a)})$, the root node of H and H_r are equal, thus $H_r \subseteq H$. 2) Suppose $H_r \subseteq T$ holds for all the parents of a node $B_r \in H_r$. There must exist a node $B \in H$ s.t. B equals to B_r. If B_r is *splittable* ($|B_r| > t$ and $B_r.upper - B_r.lower > \epsilon$), B must also be splittable because $|B| \geq |B_r| > t$ and $B.upper = B_r.upper, B.lower = B_r.lower$. Meanwhile, the split point $(B.upper - B.lower)/2 = (B_r.upper - B_r.lower)/2$. Hence, the left and right child of B and B_r are equal, thus $H_r \subseteq H$. If B_r is not *splittable*, the "child nodes" of B_r satisfy $H_r \subseteq H$ because $\emptyset \subseteq H$. Therefore, $H_r \subseteq H$ is proved by induction. □

Theorem 4.2. $\textit{RemoveFromTree}(H, \mathbf{x}_d^{(a)}, t) = H_r$.

Proof. According to Lemma 4.1, $H_r \subseteq H$. There is a sub-tree H' in H s.t. H' equals H_r but has different numbers of instances in each node. After RemoveFromTree, the number of instances in H' and H_r also become the same. Finally, since the number of instances in all the leaf nodes of H_r is smaller than t, the sub-trees of corresponding equal nodes in H' will be pruned. Therefore, after the pruning, $H = H' = H_r$. □



(a) Structure of bin tree



(b) Histogram distribution of bin tree

Fig. 5. Structure of Robust Balanced Histogram

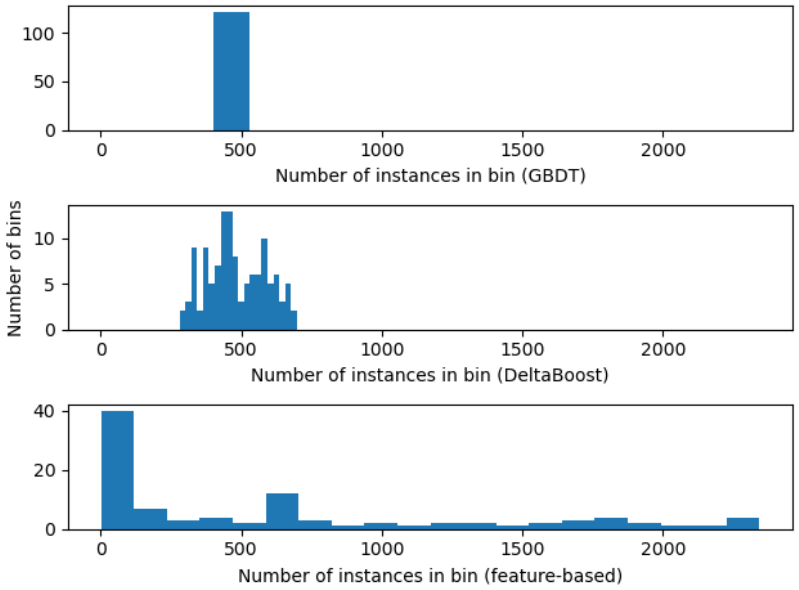


Fig. 6. Distribution of bin sizes in first feature of codrna

Algorithm 2: Build bin-tree

Data: Feature values $\mathbf{x}^{(a)} = \{x_1^{(a)}, ..., x_n^{(a)}\}$ of feature a and the range of feature values $(x_{min}^{(a)}, x_{max}^{(a)})$, max bin size t, threshold ϵ

/* A bin contains {lower, upper, left, right, #ins}, $|B|$ means the number of $x_i^{(a)}$ s.t. $B.lower \leq x_i^{(a)} < B.upper$ */

function *SplitBin*(B,m)**:**
  $B_l \leftarrow \{B.lower, m, \emptyset, \emptyset, \#\text{ins}_l\}$;
  $B_r \leftarrow \{m, B.upper, \emptyset, \emptyset, \#\text{ins}_r\}$;
  $B \leftarrow \{B.lower, B.upper, B_l, B_r, \#\text{ins}\}$; // Update child nodes
  return B_l, B_r;
function *BuildHistogram*($\mathbf{x}^{(a)}$,t,ϵ)**:**
  Split Bin $B \leftarrow \{x_{min}^{(a)}, x_{max}^{(a)}, \emptyset, \emptyset\}$;
  Tree $H \leftarrow \{B\}$;
  while $|B| > t$ **do**
    $m \leftarrow (B.upper - B.lower)/2$; // Get mean feature value
    $B_l, B_r \leftarrow$ SplitBin(B, m);
    $H \leftarrow H + \{B_l, B_r\}$;
    $B \leftarrow \arg\max_{B_i}\{|B_i| \big| B.left = \emptyset, B.right = \emptyset, B.upper - B.lower > \epsilon\}$;
  end
  prune empty bins in H;
  return H;

Algorithm 3: Remove instances from bin-tree

Data: Feature values to be deleted $\mathbf{x}_d^{(a)} = \{x_1^{(a)}, ..., x_k^{(a)}\}$ in feature a, bin-tree H, threshold t

function *RemoveFromTree*(H,$\mathbf{x}_d^{(a)}$, t)**:**
  for $x_i^{(a)} \in \mathbf{x}_d^{(a)}$ **do** // in parallel
    $B \leftarrow$ root node of H;
    while $B \neq \emptyset$ **do**
      $B.size \leftarrow B.size - 1$;
      Split value $s \leftarrow B.left.upper$; // or $B.right.lower$
      if $x_i^{(a)} < s$ **then**
        $B \leftarrow B.left$;
      else
        $B \leftarrow B.right$;
      end
    end
  end
  Prune all the bins B that $B.size < t$ by depth-first search;

4.2 Split Neighborhood

For each feature, the split value with the best gain might be slightly changed after deletion according to Theorem 3.1. To reduce such effects on the tree structure, we denote a group of neighboring split candidates in the same feature as a *split neighborhood*. The structure of split neighborhood is displayed in Fig. 7. During the tree construction, we evaluate and compare overall scores of split neighborhoods instead of gains of single split candidates. Formally, consider a split neighborhood $S_j^{(a)} = [s_j^{(a)}, ..., s_{j+k}^{(a)}]$ consisting of $k+1$ split candidates on feature a. The overall score is calculated by averaging gains of split candidates in the neighborhood as Equation 7. Each node selects the split neighborhood with the max overall score; the split candidate in this split neighborhood with the max gain value is selected as the split point.

$$\phi(S_j^{(a)}) = \frac{1}{k+1} \sum_{i=j}^{j+k} \phi(s_i^{(a)}) \tag{7}$$

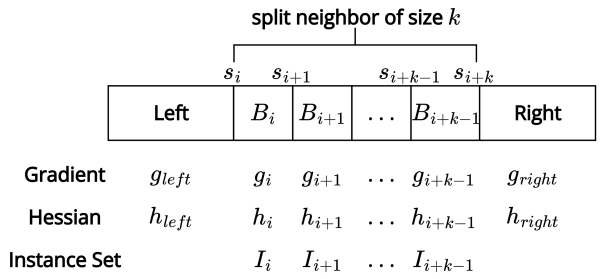


Fig. 7. Structure of split neighborhood

Besides the split values, we also store the sum of gradients and Hessians in k bins as well as that in left and right child nodes. In addition, the instances indices in the k bins are also stored in the

neighborhood to speed up deletion. This information enables instant updates of gain values in the split neighborhood.

4.3 Robust Feature Selection

Different features might have similar distributions, thus proposing split candidates with similar gains. If we always select the split candidate with the best gain as the split point, the feature split point can be easily changed after deleting a few instances. Therefore, we propose a robust approach to select the split feature as shown in Fig. 8.

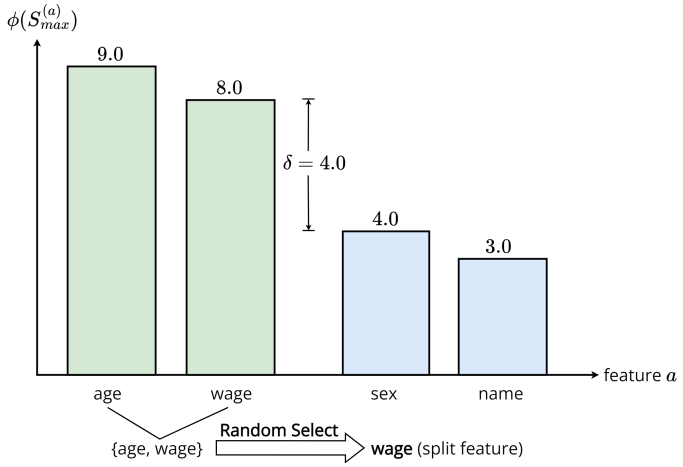


Fig. 8. Procedure of feature selection

Intuitively, we aim to find a "gap" between good and bad features. Formally, after proposing the best split neighborhood in each feature, we sort the proposed score $\phi(S_j^{(a)})$ of each feature in descending order. Then, starting from the feature with the max score, we search for a feature a s.t. $\phi(S_j^{(a)}) - \phi(S_j^{(a+1)}) > \delta$, where δ is a hyperparameter measuring the gap size. Finally, from all the features has no smaller scores than a, i.e., $\{\phi(S_j^{(i)})\}_{i=1}^{a}$, we randomly select a split neighborhood for this node.

4.4 Robust Boosting

We adopt two techniques to reduce tree dependency: bagging and gradient quantization. Both techniques are studied [11, 28] for the efficiency of GBDT training but receive less attention in model unlearning. In DeltaBoost, we find these techniques can improve the robustness of boosting.

Bagging. The global dataset $\mathbf{X} = \{\mathbf{x}_i\}_{i=1}^{n}$ is divided into ξ disjoint datasets $\{\mathbf{X}_1, ..., \mathbf{X}_\xi\}$ by instance such that $\mathbf{X} = \mathbf{X}_1 \cup ... \cup \mathbf{X}_\xi$. In iteration t, the tree T_t is trained by $\mathbf{X}_t$ and the corresponding labels $\mathbf{y}_t$ instead of the global dataset. In the inference, all the trees are used for prediction.

Gradient Quantization. DeltaBoost adopts random quantization proposed in [28]. The random quantization function $q(x)$ is formally stated as follows.

$$q(x) = \begin{cases} \lceil x \rceil, & w.p. \quad x - \lfloor x \rfloor \\ \lfloor x \rfloor, & w.p. \quad \lceil x \rceil - x \end{cases} \tag{8}$$

This function provides good performance mainly because $\mathbb{E}[q(x)] = x$. Let δ_g and δ_h be the quantized interval length of gradients and Hessians, respectively. Each tree is trained with quantized gradients $\tilde{g_i} = q(g_i/\delta_g) \cdot \delta_g,\ \tilde{h_i} = q(h_i/\delta_h) \cdot \delta_h$ instead of $g_i,\ h_i$.

5 ALGORITHM AND METRIC

In this section, we introduce the training and unlearning algorithms of DeltaBoost and the metric of unlearning. We design a robust boosting algorithm to train multiple trees with bagging and random quantization. The training and unlearning algorithm are summarized in Algorithm 4 and Algorithm 5, and a metric used to quantify the effect of unlearning described in Section 5.3.

5.1 Training

The training process is summarized in Algorithm 4. In each tree, we recursively split nodes from the root node until the sum of Hessians in a node is lower than h_{min} (lines 2-4). When splitting each node, a balanced robust histogram $\mathbf{H}$ is constructed according to Section 4.1 (line 5). Then, the score of each candidate split neighborhood in $\mathbf{H}$ is calculated according to Equation 7 and Equation 2 (line 6). Each feature proposes a split neighborhood with the max score (line 7). From these proposed split neighborhoods, the best split neighborhood S^* is selected according to Section 4.3 (line 8). Finally, this node is split with the best split candidate in S^* (lines 9-10).

The boosting process largely follows GBDT. The only difference is that DeltaBoost performs bagging (line 17) and gradient quantization (line 18) according to Section 4.4 before building a new tree.

5.2 Unlearning

The process of unlearning is summarized in Algorithm 5. For each tree, we update the gradients and Hessians of each node in parallel with the removed gradients (lines 2-6). The existence of removed instances in each node can either be stored during the training or calculated by inference with $\mathbf{x}_d$ during the deletion. After the update, the best split candidate for some split neighborhoods might change. Each change causes the instances between the previous and current split point to shift from left child to right child (or from right to left). Therefore, we merge these shifted gradients and Hessians from previous layers (lines 10-12) before updating the best split candidate (line 13) and propagating the shifted gradients to the deeper layers (lines 14-17). This approach ensures exact deletion if the split candidates of all the retrained new nodes are still within the original split neighborhood.

5.3 Metric for Unlearning

In this subsection, we propose a metric to compare M_d and M_r. The most direct approach is to compare the model parameters of M_d and M_r. Although this approach is adopted in some neural-network models [20, 31], it is infeasible for tree-based models since trees may differ in not only parameter values but also structures. Therefore, the evaluation of the existing studies [7, 27] of tree-based models usually rely on the comparison between the accuracy of M_d and M_r. Specifically, these studies assume M_d and M_r with the same accuracy on $\mathbf{D}_r$, $\mathbf{D}_d$, and a test dataset $\mathbf{D}_{test}$ are the same model, leading to a clean deletion. Nonetheless, the same accuracy does not imply the same prediction for every instance because 1) raw predicted scores are rounded before comparing, 2) predictions of all instances are averaged before comparing. Considering the two factors of information loss, we propose a more sensitive metric to compare M_d and M_r.

Since many tree-based models are trained in probabilistic algorithms (e.g., random forest), we consider the learned model and the model outputs as random variables. Our metric calculates the

Algorithm 4: Training of DeltaBoost

Data: Data $\mathbf{x}$, labels $\mathbf{y}$, threshold t, ϵ, min node weights h_{min}, max depth κ of trees

function *TrainTree*($\mathbf{x}$,$\mathbf{g}$,$\mathbf{h}$)**:**
- **if** $\sum_{h_i \in \mathbf{h}} h_i < h_{min}$ ***or*** *current depth* $> \kappa$ **then**
  - **return** Leaf with weight w_T calculated by Equation 3;
- **end**
- $\mathbf{H} \leftarrow$ BuildHistogram$(\mathbf{x}^{(a)}, t, \epsilon)$ for each feature a ;
- $\Phi_0 \leftarrow$ calculate gain $\phi_0(S_i^{(a)})$ for each split neighborhood $S_i^{(a)} \in H$ with $\mathbf{g}, \mathbf{h}$ by Equation 7 and Equation 2 ;
- $\{\phi_0^{(a)*}\}_{a=1}^m \leftarrow \{\max_i \phi_0(S_i^{(a)})\}_{a=1}^m$;
- $S^*, \phi_0^* \leftarrow$ propose the split neighborhood from $\{\phi_0^{(a)*}\}_{a=1}^m$ according to Section 4.3 ;
- $s^* \leftarrow$ split point with max gain in S^* ;
- $(\mathbf{x}_l, \mathbf{g}_l, \mathbf{h}_l), (\mathbf{x}_r, \mathbf{g}_r, \mathbf{h}_r) \leftarrow$ split $\mathbf{x}, \mathbf{g}, \mathbf{h}$ at s^* ;
- $T_l \leftarrow$ TrainTree$(\mathbf{x}_l, \mathbf{g}_l, \mathbf{h}_l)$, $T_r \leftarrow$ TrainTree$(\mathbf{x}_r, \mathbf{g}_r, \mathbf{h}_r)$;
- **return** node split at s^* and subtrees T_l, T_r ;

function *TrainBoost*($\mathbf{x}$, $\mathbf{y}$)**:**
- $M_0 \leftarrow \emptyset$, $L^{(0)} \leftarrow$ Equation 1 with $\mathbf{y}$;
- **for** $t \leftarrow 1$ *to* t_{max} **do**
  - $\mathbf{g}_t, \mathbf{h}_t \leftarrow$ calculated from $L^{(t)}$;
  - $\mathbf{x}_t, \mathbf{g}'_t, \mathbf{h}'_t \leftarrow$ randomly sampled from $\mathbf{x}, \mathbf{g}_t, \mathbf{h}_t$;
  - $\tilde{\mathbf{g}}'_t, \tilde{\mathbf{h}}'_t \leftarrow$ randomly quantized $\mathbf{g}'_t, \mathbf{h}'_t$ by Equation 8 ;
  - $T_t \leftarrow$ TrainTree$(\mathbf{x}_t, \tilde{\mathbf{g}}'_t, \tilde{\mathbf{h}}'_t)$;
  - $M_t \leftarrow M_{t-1} \cup \{T_t\}$;
  - $L^{(t)} \leftarrow$ update loss function with M_t and $\mathbf{y}$ (Equation 1);
- **end**
- **return** $M_{t_{max}}$;

instance-level squared Hellinger distance [19] on the raw predicted scores of $M_d(\mathbf{D})$ and $M_r(\mathbf{D})$ and scales it with the squared Euclidean distances between their expected values. Finally, these per-instance scores are averaged. Formally, the difference between M_d and M_r on dataset $\mathbf{D}$ is evaluated by Equation 9.

$$H^2(M_r, M_d; \mathbf{D}) = \left\| \mathbb{1}_{n\times 1} - \int \sqrt{M_r(x; \mathbf{D}) M_d(x; \mathbf{D})}\, dx \right\|_1 \tag{9}$$

We denote $H^2(M_r, M_d; \mathbf{D})$ as the *forgetfulness* $\mathcal{F}(M, \mathbf{D})$ of model M on dataset D. For $f_{unlearn}$ that performs exact unlearning, it holds $H^2(M_r, M_d; \mathbf{D}) = 0$. For $f_{unlearn}$ that does nothing, it holds $H^2(M_r, M_d; \mathbf{D}) = H^2(M_r, M; \mathbf{D})$. Smaller forgetfulness indicates cleaner deletion. This metric is much more sensitive than accuracy for eliminating both factors of information loss, which has also been demonstrated in our experiments.

Estimation of Hellinger distance. We estimate the Hellinger distance between the two models by sampling. Specifically, on the same arbitrary dataset D, both models M_r and M_d are trained for ν times with different random seeds. Then, each instance has ν predictions as a sampling from the prediction's probabilistic density distribution (PDF). Finally, we estimate the integral of Equation 9 by quadrature. Specifically, let the range of the output of $M_r(x; \mathbf{D})$ and $M_d(x; \mathbf{D})$ be

Algorithm 5: Unlearning from DeltaBoost

Data: Data to be unlearned $\mathbf{x}_d, \mathbf{y}_d$, original model M, number of trees $t_{max} \in M$, quantized gradients $\tilde{\mathbf{g}}, \tilde{\mathbf{h}} = \{\tilde{\mathbf{g}}^{(t)}, \tilde{\mathbf{h}}^{(t)}\}_{t=1}^{t_{max}}$ in each tree T_t, threshold t, ϵ, min node weights h_{min}

function *UnlearnTree(()*$T, \mathbf{x}_d, \mathbf{g}, \mathbf{h}, \mathbf{g}_d, \mathbf{h}_d$*)*:
  /* Remove instances from nodes */
  for *node* $N_j \in T$ ***in parallel*** **do**
    for $g_i, h_i \in \{\mathbf{g}_d, \mathbf{h}_d\}$ ***in parallel*** **do**
      Update gradients of N_j with g_i, h_i s.t. $\mathbf{x}_i \in N_j$;
    end
  end
  /* Recalculate the split point in split neighborhoods */
  $\mathbf{g}'[N_i] \leftarrow \emptyset, \mathbf{h}'[N_i] \leftarrow \emptyset$ **for each** $N_i \in T$;
  for $d \leftarrow 1$ *to Depth*(T) **do**
    for *node* $N_j \in$ *all nodes in depth* d ***in parallel*** **do**
      for $g'_i, h'_i \in \{\mathbf{g}'[N_j], \mathbf{h}'[N_j]\}$ ***in parallel*** **do**
        Update gradients in n_j;
      end
      $s_j^{*\prime} \leftarrow$ update split point in N_j ;
      $N_l, N_r \leftarrow$ left and right node of N_j;
      $\mathbf{g}'_l, \mathbf{h}'_l \leftarrow$ gradients between s^* and $s^{*\prime}$;
      $\mathbf{g}'_r \leftarrow -\mathbf{g}'_l, \mathbf{h}'_r \leftarrow -\mathbf{h}'_l$;
      $\mathbf{g}'[N_l] \leftarrow \mathbf{g}'[N_j] \cup \mathbf{g}'_l, \mathbf{g}'[N_r] \leftarrow \mathbf{g}'[N_j] \cup \mathbf{g}'_r$;
    end
  end
  return T;
function *UnlearnBoost(*$M, \mathbf{x}_d, \tilde{\mathbf{g}}, \tilde{\mathbf{h}}, \tilde{\mathbf{g}}_d, \tilde{\mathbf{h}}_d$*)*:
  for *tree* $T_t \in M$ ***in parallel*** **do**
    $\tilde{\mathbf{g}}^{(t)}, \tilde{\mathbf{h}}^{(t)}, \tilde{\mathbf{g}}_d^{(t)}, \tilde{\mathbf{h}}_d^{(t)} \leftarrow$ get used gradients in T_t from $\tilde{\mathbf{g}}, \tilde{\mathbf{h}}, \tilde{\mathbf{g}}_d, \tilde{\mathbf{h}}_d$;
    $T_t \leftarrow$ UnlearnFromTree$(T, \mathbf{x}_d, \tilde{\mathbf{g}}', \tilde{\mathbf{h}}', \tilde{\mathbf{g}}'_d, \tilde{\mathbf{h}}'_d)$;
  end
  return M

$[o_{min}, o_{max}]$. Dividing the range of outputs into β small intervals $\rho_1, ..., \rho_\beta$ of width γ, supposed τ_i^r is the frequency of $M_r(x; \mathbf{D})$ falling into interval ρ_i, τ_i^d is the frequency of $M_d(x; \mathbf{D})$ falling into interval ρ_i, the Hellinger distance is estimated by Equation 10.

$$H^2(M_r, M_d; \mathbf{D}) \approx \left\| \mathbb{1}_{n\times 1} - \frac{o_{max} - o_{min}}{\beta} \sum_{i=1}^{\beta} \sqrt{\tau_i^r \tau_i^d} \right\|_1 \tag{10}$$

6 EXPERIMENT

6.1 Experimental Settings

Datasets. The experiments adopt five datasets, including three binary classification datasets and two regression datasets. The details of these datasets are summarized in Table 3. For `cadata` and `covtype` with no public test set provided, we randomly select 20% instances as the test set.

Metrics. In the evaluation of unlearning, we present the results of two metrics: 1) forgetfulness proposed in Section 5.3 as well as the Hellinger distance between M and M_d for reference; 2) *error* of M, M_r, and M_d. For classification tasks, the error is defined as 1−Accuracy; for regression tasks, the error is defined as the root mean square error (RMSE). In the evaluation of performance, we present the error on each dataset. In the evaluation of efficiency, we present the running time in seconds.

Hardware and Implementation. Experiments are run on an Intel(R) Xeon(R) Gold 6248R CPU @ 3.00GHz with 380GB of RAM. DeltaBoost is implemented based on ThunderGBM [30] with C++ 17 and compiled by g++ 10.3 with -O3 optimization.

Baselines. In evaluating unlearning, since no studies of unlearning GBDT have been proposed, we compare a single tree of DeltaBoost with a single tree of HedgeCut [27], and DaRE [7]. Both HedgeCut and DaRE are not evaluated on `cadata` and `msd` since they do not support regression tasks. We evaluate performance by comparing DeltaBoost with existing tree-based models, including ThunderGBM-CPU [22], XGBoost [10], sklearn decision tree, and sklearn random forest. In the evaluation efficiency, the training time and deletion time of DeltaBoost is compared with the retraining time of XGBoost, sklearn GBDT, and ThunderGBM-CPU.

Hyperparameters. In all the experiments, we choose the max depth of each tree in {5,7}. The learning rate of DeltaBoost is selected from $\{0.3, 1\}$. The regularization parameter λ is set to zero. The number of quantized intervals is selected from $\{8, 16, 32\}$. We set $t_{max} = 10, \xi = 2$ for the training of DeltaBoost. In the estimation of forgetfulness, we set $\beta = 50, \nu = 100$.

Table 3. Basic information of datasets ("bin-cls" means binary classification; "reg" means regression.)

Dataset	Task	#instances	% of train	#attributes	Ref
codrna	bin-cls	331,152	17.98%	8	[2]
covtype	bin-cls	581,012	80.00%	54	[3]
gisette	bin-cls	7,000	85.71%	5,000	[4]
cadata	reg	20,640	80.00%	8	[1]
msd	reg	515,345	89.98%	90	[6]

6.2 Unlearning

In this subsection, we evaluate how clean samples are unlearned in DeltaBoost. In the first part, we compare the forgetfulness of single-tree DeltaBoost with single-tree DaRE and HedgeCut. In the second part, we present the forgetfulness of DeltaBoost with multiple trees.

A Single Tree. The test error of each approach on M, M_d, and M_r, is presented in Table 4. The forgetfulness of each approach is presented in Table 5. From the two tables, we observe that DeltaBoost has forgotten the deleted instances more cleanly than HedgeCut and DRAT in a single tree. In Table 4, for DeltaBoost, the error of the deleted model M_d is closer to the error of the removed model M_r than the error of the original model M, which implies that M_d is more similar to M_r than M. This observation is further validated by our proposed metric in Table 5. The squared Hellinger distances between M_r and M_d are reduced below 0.01 by DeltaBoost in all the datasets despite the large distance between M and M_r. Nonetheless, similar performance is not observed from baselines. Although DaRE theoretically guarantees exact deletion, its large amount of noise causes large variance in forgetfulness and significantly harms accuracy as shown in Table 4. For

HedgeCut, the performance of M_d is more similar to M than M_r in Table 4 and both distances $H^2(M_r, M; \mathbf{D}_{test}), H^2(M_r, M_d; \mathbf{D}_{test})$ are very small in Table 5. This indicates that HedgeCut tends to preserve the same tree structure in the retraining instead of developing an effective deleting algorithm. Despite its strong privacy guarantee, such an excessively robust structure harms the model's accuracy.

Table 4. The test error on M, M_d, and M_r (single tree)

Dataset	Model	Error on D_{test}					
		Remove 0.1%			Remove 1%		
		DaRE	HedgeCut	DeltaBoost	DaRE	HedgeCut	DeltaBoost
codrna	M	0.1703±0.0576	0.0908±0.0041	0.0874±0.0002	0.1703±0.0576	0.0931±0.0050	0.0873±0.0005
	M_d	0.1685±0.0545	0.0908±0.0041	0.0874±0.0002	0.1827±0.0642	0.0935±0.0050	0.0873±0.0005
	M_r	0.1755±0.0575	0.0902±0.0047	0.0873±0.0002	0.1784±0.0634	0.0910±0.0046	0.0872±0.0007
covtype	M	0.2870±0.0463	0.1680±0.0035	0.2611±0.0001	0.2870±0.0463	0.1694±0.0043	0.2610±0.0001
	M_d	0.2874±0.0466	0.1681±0.0035	0.2611±0.0001	0.2884±0.0465	0.1698±0.0043	0.2611±0.0001
	M_r	0.2866±0.0465	0.1677±0.0039	0.2611±0.0001	0.2862±0.0480	0.1684±0.0041	0.2611±0.0001
gisette	M	0.1055±0.0135	0.3518±0.0214	0.0736±0.0017	0.1055±0.0135	0.3732±0.0245	0.0788±0.0040
	M_d	0.1053±0.0134	0.3520±0.0215	0.0736±0.0017	0.1052±0.0139	0.3735±0.0245	0.0788±0.0040
	M_r	0.1064±0.0144	0.3531±0.0234	0.0736±0.0017	0.1068±0.0142	0.3495±0.0207	0.0788±0.0037
cadata	M	-	-	0.1557±0.0034	-	-	0.1643±0.0066
	M_d	-	-	0.1558±0.0034	-	-	0.1644±0.0065
	M_r	-	-	0.1558±0.0034	-	-	0.1644±0.0065
msd	M	-	-	0.1009±0.0003	-	-	0.1009±0.0003
	M_d	-	-	0.1009±0.0003	-	-	0.1009±0.0003
	M_r	-	-	0.1009±0.0003	-	-	0.1009±0.0003

Table 5. The forgetfulness $\mathcal{F}(M, D_{test})$ of each approach (single tree)

Dataset	Metric	Hellinger distances $H^2(M_r, M; \mathbf{D}_{test})$ and $H^2(M_r, M_d; \mathbf{D}_{test})$					
		Remove 0.1%			Remove 1%		
		DaRE	HedgeCut	DeltaBoost	DaRE	HedgeCut	DeltaBoost
codrna	$H^2(M_r, M; \mathbf{D}_{test})$	0.0913±0.0447	0.0031±0.0047	0.0002±0.0051	0.0949±0.0514	0.0040±0.0192	0.1046±0.2984
	$H^2(M_r, M_d; \mathbf{D}_{test})$	0.0866±0.0414	0.0031±0.0047	0.0000±0.0014	0.0839±0.0490	0.0035±0.0066	0.0070±0.0515
covtype	$H^2(M_r, M; \mathbf{D}_{test})$	0.0750±0.0304	0.0034±0.0052	0.0162±0.1260	0.0921±0.0344	0.0036±0.0070	0.0232±0.1459
	$H^2(M_r, M_d; \mathbf{D}_{test})$	0.0746±0.0301	0.0034±0.0049	0.0000±0.0005	0.0933±0.0349	0.0035±0.0056	0.0001±0.0011
gisette	$H^2(M_r, M; \mathbf{D}_{test})$	0.0997±0.0573	0.0014±0.0023	0.0005±0.0018	0.0958±0.0584	0.0033±0.0045	0.0116±0.0100
	$H^2(M_r, M_d; \mathbf{D}_{test})$	0.1007±0.0572	0.0014±0.0023	0.0001±0.0006	0.0950±0.0588	0.0033±0.0045	0.0086±0.0080
cadata	$H^2(M_r, M; \mathbf{D}_{test})$	-	-	0.0036±0.0098	-	-	0.0087±0.0112
	$H^2(M_r, M_d; \mathbf{D}_{test})$	-	-	0.0015±0.0032	-	-	0.0033±0.0048
msd	$H^2(M_r, M; \mathbf{D}_{test})$	-	-	0.0041±0.0044	-	-	0.0126±0.0101
	$H^2(M_r, M_d; \mathbf{D}_{test})$	-	-	0.0028±0.0036	-	-	0.0093±0.0079

Multiple Trees. The test error of converged DeltaBoost is presented in Table 7a. The forgetfulness of converged DeltaBoost is summarized in Table 7b. As observed from Table 7a, the accuracy of M_d is closer to the accuracy of M_r than M in most cases. From Table 7b, we find the Hellinger distances between M_d and M_r, though slightly larger than the distance of a single tree, still remain

a small value (under 0.05). This indicates that DeltaBoost can also effectively remove instances in the boosting. We also notice that $H^2(M_r, M; \mathbf{D}_{test})$ is close to $H^2(M_r, M_d; \mathbf{D}_{test})$, which means DeltaBoost is very robust, thus raising a potential concern on the accuracy. This accuracy concern has been addressed in Section 6.4 by showing that DeltaBoost has comparable accuracy to existing tree-based models.

Table 6. Training time and removal time of DeltaBoost (in seconds). Speedup = training time / DeltaBoost-Remove. "Sklearn" means sklearn GBDT. "Thunder" means ThunderGBM-CPU. "DB" means DeltaBoost.

Dataset	Remove Ratio	Time (seconds)					DB-Remove Speedup v.s.		
		XGBoost	sklearn	Thunder	DB-Train	DB-Remove	XGBoost	sklearn	Thunder
codrna	0.1%	0.238	1.050	0.502	3.475 ±0.047	0.078 ±0.020	3.05×	13.46×	6.44×
	1%	0.317	1.055	0.563	3.362 ±0.054	0.076 ±0.023	4.17×	13.88×	7.41×
covtype	0.1%	4.093	21.556	3.112	47.070 ±1.034	0.679 ±0.560	6.03×	31.75×	4.58×
	1%	4.030	21.284	3.593	47.114 ±0.896	1.197 ±1.837	3.37×	17.78×	3.00×
gisette	0.1%	1.233	14.699	9.820	22.853 ±2.416	0.095 ±0.015	12.98×	154.73×	103.37×
	1%	1.213	14.451	10.011	22.538 ±2.159	0.071 ±0.004	17.08×	203.54×	141×
cadata	0.1%	0.209	0.484	0.243	1.021 ±0.023	0.056 ±0.053	3.73×	8.64×	4.34×
	1%	0.219	0.480	0.260	0.952 ±0.033	0.042 ±0.029	5.21×	11.43×	6.19×
msd	0.1%	2.547	300.637	7.415	45.369 ±1.264	0.334 ±0.048	7.63×	900.11×	22.2×
	1%	2.516	297.592	7.981	45.687 ±0.947	0.434 ±0.073	5.8×	685.7×	18.39×
Geomean							5.88×	53.78×	12.65×

Table 7. The test error and forgetfulness of DeltaBoost (multiple trees)

(a) The test error on M, M_d, and M_r

Dataset	Model	Error on D_{test}	
		Remove 0.1%	Remove 1%
codrna	M	0.0616±0.0010	0.0617±0.0009
	M_d	0.0617±0.0010	0.0618±0.0010
	M_r	0.0618±0.0011	0.0618±0.0011
covtype	M	0.2265±0.0070	0.2265±0.0070
	M_d	0.2264±0.0070	0.2265±0.0069
	M_r	0.2265±0.0066	0.2375±0.0081
gisette	M	0.0519±0.0041	0.0496±0.0045
	M_d	0.0520±0.0041	0.0496±0.0045
	M_r	0.0519±0.0042	0.0499±0.0049
cadata	M	0.1269±0.0052	0.1396±0.0067
	M_d	0.1273±0.0051	0.1401±0.0067
	M_r	0.1271±0.0053	0.1401±0.0070
msd	M	0.1040±0.0006	0.1121±0.0001
	M_d	0.1040±0.0006	0.1133±0.0220
	M_r	0.1041±0.0006	0.1121±0.0001

(b) The forgetfulness of DeltaBoost $\mathcal{F}(M, D_{test})$

Dataset	Metric	Hellinger distances	
		Remove 0.1%	Remove 1%
codrna	$H^2(M_r, M; \mathbf{D}_{test})$	0.0121±0.0095	0.0094±0.0082
	$H^2(M_r, M_d; \mathbf{D}_{test})$	0.0121±0.0095	0.0094±0.0081
covtype	$H^2(M_r, M; \mathbf{D}_{test})$	0.0124±0.0097	0.0408±0.0350
	$H^2(M_r, M_d; \mathbf{D}_{test})$	0.0124±0.0097	0.0407±0.0348
gisette	$H^2(M_r, M; \mathbf{D}_{test})$	0.0204±0.0111	0.0281±0.0147
	$H^2(M_r, M_d; \mathbf{D}_{test})$	0.0203±0.0113	0.0278±0.0145
cadata	$H^2(M_r, M; \mathbf{D}_{test})$	0.0240±0.0175	0.0249±0.0160
	$H^2(M_r, M_d; \mathbf{D}_{test})$	0.0238±0.0159	0.0249±0.0148
msd	$H^2(M_r, M; \mathbf{D}_{test})$	0.0187±0.0102	0.0247±0.0126
	$H^2(M_r, M_d; \mathbf{D}_{test})$	0.0188±0.0102	0.0247±0.0125

6.3 Efficiency

Time efficiency. The time cost for training and deletion are presented in Table 6. We compare the retraining time of GBDT on the remained dataset, the training time of DeltaBoost on the whole dataset, and the total removal time of DeltaBoost. To enable fast data removal instead of retraining

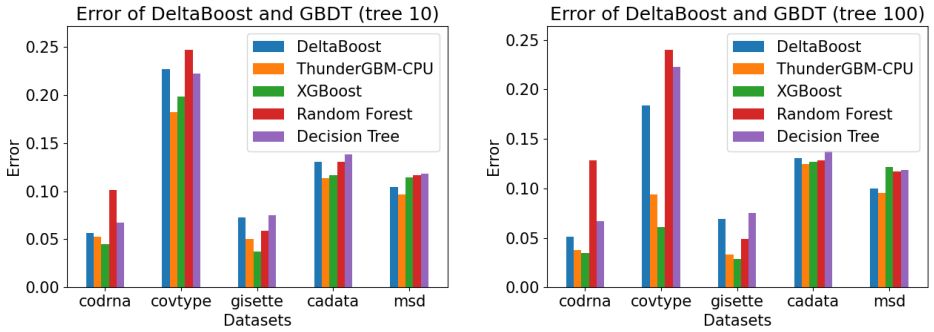


Fig. 9. Performance of DeltaBoost and tree-based models

from scratch, DeltaBoost introduces acceptable additional training costs. The removal algorithm has 12.65× speedup compared to ThunderGBM-CPU on average. The speedup is more significant on datasets on high dimension datasets (e.g., `gisette`). DeltaBoost is much more efficient than retraining and applies to real-world machine learning applications such as online recommendations.

Memory efficiency. We evaluate the peak memory usage during training DeltaBoost and ThunderGBM-CPU in Table 8. DeltaBoost requires 6.80× memory overhead on average to enable fast deletion.

Table 8. Memory cost of DeltaBoost (10 trees) in Megabytes

Dataset	Data	DeltaBoost	ThunderGBM-CPU	Overhead
codrna	3.9	590	92	6.41×
covtype	120	20,546	1,373	14.96×
gisette	158	23,718	6,420	3.69×
cadata	1.4	148	49	3.02×
msd	407	29,941	2,199	13.62×
		Geomean		6.80×

6.4 Accuracy

In this subsection, we compare the accuracy of DeltaBoost after unlearning with retraining on the remained dataset using popular tree-based machine learning models, including GBDT and random forests. We set the removal ratio to 0.1%. The error of these approaches is plotted in Fig. 9. As observed from Fig. 9, despite slight performance loss compared to GBDT models, DeltaBoost has comparable performance to existing tree-based models.

6.5 Ablation Study

In this subsection, we perform ablation studies to investigate the effect of components and hyperparameters in DeltaBoost. Since forgetfulness is more sensitive than accuracy as elaborated in Section 5.3, we adopt forgetfulness to evaluate the effectiveness of deletion.

Number of Trees. The number of trees affects both the accuracy and forgetfulness of DeltaBoost. For accuracy, in Fig. 9, we observe that training more iterations above 10 has only small improvements in accuracy (except on covtype). For forgetfulness, in Fig. 10, training more trees at first leads

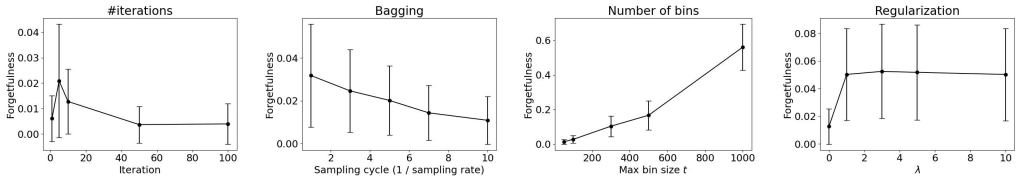


Fig. 10. Effect of different hyperparameters on codrna dataset

to worse forgetfulness because the error accumulates across trees. Nonetheless, after DeltaBoost converges, training more trees results in better forgetfulness because changes in a single tree have less effect on the final prediction.

Bagging. The effect of bagging on the forgetfulness is presented in Fig. 10. It can be observed that a larger sampling cycle leading to a smaller sampling rate results in a smaller distance between M_r and M_d. This observation demonstrates that bagging can improve forgetfulness.

Max Bin Size. The number of bins, which differs by features, is controlled by max bin size in DeltaBoost. A larger max bin size means a smaller number of bins. In Fig. 10, we observe that a larger max bin size leads to worse forgetfulness. This is because the shift in a split value may affect more instances.

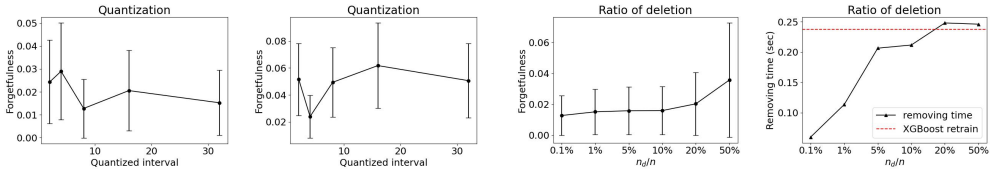
Regularization. The effect of regularization is presented in Fig. 10. A small λ leads to the best forgetfulness because both Theorem 3.1 and the quantization technique assume a negligible λ. Meanwhile, a large λ does not dramatically affect forgetfulness.

Deletion Rate. We increase the deletion rate above 1% and record the forgetfulness and the total deletion time in Fig. 11b. It can be observed that the mean and variance of Hellinger distance increase rapidly when n_d/n is above 10%. Meanwhile, a larger removal ratio also incurs more removal time. When the removal ratio exceeds 10%, removal is more expensive than retraining. This indicates that DeltaBoost is more efficient than retraining when the removing ratio is relatively small. If the removing ratio exceeds a certain limit, DeltaBoost may no longer be applicable for efficient data deletion because the removal time may be longer than retraining. Meanwhile, the quality of removal may not be guaranteed since Theorem 3.1 does not hold anymore. Therefore, when the limit is reached, retraining of DeltaBoost is recommended to support further deletion requests.

Quantization. The effect of quantization on the forgetfullness is presented in Fig. 11a. Compared to other techniques, gradient quantization has some but less contribution to forgetfulness. Over-quantization can lead to worse forgetfulness in DeltaBoost.

7 RELATED WORK

As the concern on data privacy increases, various machine unlearning approaches have been developed to efficiently delete learned data records from models. Some early approaches [8, 26, 29] are designed for linear models. Though the linearity of models makes data removal convenient, it also results in poor performance in fitting complicated data distributions. Therefore, efficient unlearning algorithms are developed in non-linear models including nearest-neighbors methods [26], k-means [12], support vector machines [9], collaborative filtering [26], and neural networks [13, 14, 23, 31]. Traditional tree-based models, which are widely adopted because of their explainability and efficiency, are usually sensitive to data removals. Hence, robust variants of traditional tree-based



(a) Effect of quantization and removing ratio on forgetfulness (left: codrna, right: msd)

(b) Effect of removal ratio on forgetfulness and removal time

Fig. 11. Effect of quantization and removal ratio

models are proposed to enable efficient deletion. Based on random forest, DaRE [7] constructs a data removal-enabled forest by selecting random split features and thresholds in shallow nodes. Based on extremely randomized trees, HedgeCut [27] stores pre-trained sub-trees for non-robust nodes during the training to enable efficient data removal. Nonetheless, efficient unlearning algorithms for GBDT [10], which is a popular tree-based algorithm with high efficiency and performance, have not been explored yet.

Existing unlearning approaches have two categories: exact unlearning [7, 8, 12, 26] and approximate unlearning [13, 14, 16, 20, 23, 27, 31]. Exact unlearning guarantees that the deleted model M_d and the retrained model M_d are exactly the same [8, 12, 26] or has the same distribution [7]. Such approaches achieve clean deletion, thus perfect data privacy. Nevertheless, these approaches either apply to a limited set of machine learning models or have poor performance due to random noise. Hence, most unlearning approaches focusing on gradient-descent models [13, 14, 20, 23, 31] (e.g., neural networks) adopt approximate unlearning that incurs less performance loss. In this study focusing on GBDT which performs gradient descent with decision trees, we also aim to explore approximate unlearning algorithms to achieve better model performance.

8 CONCLUSION

In this paper, we theoretically analyze the effect of random removing instances on the tree structure of GBDT. With the conclusion that the small fraction of removal only causes a slight shift of the best split gain, we propose split neighborhood to enhance the robustness of decision trees. We propose a robust feature selection technique to prevent features with close gain values from competing. Furthermore, we adopt gradient quantization and bagging during the training to reduce the dependency across trees. In summary, we propose a robust GBDT-like machine learning model and the corresponding efficient unlearning algorithm. Our experiments on five datasets demonstrate that DeltaBoost enables efficient and effective unlearning while providing competitive performance to existing tree-based ML models.

ACKNOWLEDGMENTS

This research/project is supported by the National Research Foundation, Singapore under its AI Singapore Programme (AISG Award No: AISG2-TC-2021-002). Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of National Research Foundation, Singapore. We thank the AMD Heterogeneous Accelerated Compute Clusters (HACC) program (formerly known as the XACC program - Xilinx Adaptive Compute Cluster program) for the generous hardware donation.

REFERENCES

[1] Cadata dataset. https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/regression.html#cadata. Accessed: 2022-10-02.

[2] Codrna dataset. https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html#cod-rna. Accessed: 2022-10-02.
[3] Covertype data set. https://archive.ics.uci.edu/ml/datasets/covertype. Accessed: 2022-10-02.
[4] Gisette data set. https://archive.ics.uci.edu/ml/datasets/Gisette. Accessed: 2022-10-02.
[5] Approximations for mean and variance of a ratio, 2022. Accessed: 2022-10-01.
[6] Thierry Bertin-Mahieux, Daniel P.W. Ellis, Brian Whitman, and Paul Lamere. The million song dataset. In *Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR 2011)*, 2011.
[7] Jonathan Brophy and Daniel Lowd. Machine unlearning for random forests. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 1092–1104. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/brophy21a.html.
[8] Yinzhi Cao and Junfeng Yang. Towards making systems forget with machine unlearning. In *2015 IEEE Symposium on Security and Privacy*, pages 463–480. IEEE, 2015.
[9] Gert Cauwenberghs and Tomaso Poggio. Incremental and decremental support vector machine learning. *Advances in neural information processing systems*, 13, 2000.
[10] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.
[11] Yasser Ganjisaffar, Rich Caruana, and Cristina Videira Lopes. Bagging gradient-boosted trees for high precision, low variance ranking models. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, pages 85–94, 2011.
[12] Antonio Ginart, Melody Guan, Gregory Valiant, and James Y Zou. Making ai forget you: Data deletion in machine learning. *Advances in neural information processing systems*, 32, 2019.
[13] Aditya Golatkar, Alessandro Achille, and Stefano Soatto. Eternal sunshine of the spotless net: Selective forgetting in deep networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9304–9312, 2020.
[14] Aditya Golatkar, Alessandro Achille, and Stefano Soatto. Forgetting outside the box: Scrubbing deep networks of information accessible from input-output observations. In *European Conference on Computer Vision*, pages 383–398. Springer, 2020.
[15] Eric Goldman. An introduction to the california consumer privacy act (ccpa). *Santa Clara Univ. Legal Studies Research Paper*, 2020.
[16] Chuan Guo, Tom Goldstein, Awni Hannun, and Laurens Van Der Maaten. Certified data removal from machine learning models. *arXiv preprint arXiv:1911.03030*, 2019.
[17] John T Hancock and Taghi M Khoshgoftaar. Catboost for big data: an interdisciplinary review. *Journal of big data*, 7 (1):1–45, 2020.
[18] William V Harper. Statistics: Theory and methods, 1991.
[19] Ernst Hellinger. Neue begründung der theorie quadratischer formen von unendlichvielen veränderlichen. *Journal für die reine und angewandte Mathematik*, 1909(136):210–271, 1909.
[20] Zachary Izzo, Mary Anne Smart, Kamalika Chaudhuri, and James Zou. Approximate data deletion from machine learning models. In *International Conference on Artificial Intelligence and Statistics*, pages 2008–2016. PMLR, 2021.
[21] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, 30, 2017.
[22] Qinbin Li, Yanzheng Cai, Yuxuan Han, Ching Man Yung, Tianyuan Fu, and Bingsheng He. Fedtree: A fast, effective, and secure tree-based federated learning system. https://github.com/Xtra-Computing/FedTree/blob/main/FedTree_draft_paper.pdf, 2022.
[23] Seth Neel, Aaron Roth, and Saeed Sharifi-Malvajerdi. Descent-to-delete: Gradient-based methods for machine unlearning. In *Algorithmic Learning Theory*, pages 931–962. PMLR, 2021.
[24] Athanasios Papoulis. *Probability and statistics*. Prentice-Hall, Inc., 1990.
[25] Eugenia Politou, Efthimios Alepis, and Constantinos Patsakis. Forgetting personal data and revoking consent under the gdpr: Challenges and proposed solutions. *Journal of cybersecurity*, 4(1):tyy001, 2018.
[26] Sebastian Schelter. "amnesia"-a selection of machine learning models that can forget user data very fast. *CIDR*, 2020.
[27] Sebastian Schelter, Stefan Grafberger, and Ted Dunning. Hedgecut: Maintaining randomised trees for low-latency machine unlearning. In *Proceedings of the 2021 International Conference on Management of Data*, SIGMOD '21, page 1545–1557, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450383431. doi: 10.1145/3448016.3457239. URL https://doi.org/10.1145/3448016.3457239.
[28] Yu Shi, Guolin Ke, Zhuoming Chen, Shuxin Zheng, and Tie-Yan Liu. Quantized training of gradient boosting decision trees. *Advances in neural information processing systems*, 2022.
[29] Cheng-Hao Tsai, Chieh-Yen Lin, and Chih-Jen Lin. Incremental and decremental training for linear classification. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 343–352, 2014.

[30] Z. Wen, J. Shi, B. He, J. Chen, K. Ramamohanarao, and Q. Li. Exploiting gpus for efficient gradient boosting decision tree training. *IEEE Transactions on Parallel and Distributed Systems*, 30(12):2706–2717, 2019.

[31] Yinjun Wu, Edgar Dobriban, and Susan Davidson. Deltagrad: Rapid retraining of machine learning models. In *International Conference on Machine Learning*, pages 10355–10366. PMLR, 2020.

[32] Zhaomin Wu, Junhui Zhu, Qinbin Li, and Bingsheng He. Technical report of deltaboost: Gradient boosting decision trees with efficient machine unlearning. https://github.com/Xtra-Computing/DeltaBoost/blob/main/DeltaBoost_Technical_Report.pdf, 2022.

Received October 2022; revised January 2023; accepted February 2023