Practical Federated Gradient Boosting Decision Trees

Qinbin Li,[1] **Zeyi Wen,**[2] **Bingsheng He**[1]
[1]National University of Singapore
[2]The University of Western Australia
{qinbin, hebs}@comp.nus.edu.sg, zeyi.wen@uwa.edu.au

Abstract

Gradient Boosting Decision Trees (GBDTs) have become very successful in recent years, with many awards in machine learning and data mining competitions. There have been several recent studies on how to train GBDTs in the federated learning setting. In this paper, we focus on horizontal federated learning, where data samples with the same features are distributed among multiple parties. However, existing studies are not efficient or effective enough for practical use. They suffer either from the inefficiency due to the usage of costly data transformations such as secret sharing and homomorphic encryption, or from the low model accuracy due to differential privacy designs. In this paper, we study a practical federated environment with relaxed privacy constraints. In this environment, a dishonest party might obtain some information about the other parties' data, but it is still impossible for the dishonest party to derive the actual raw data of other parties. Specifically, each party boosts a number of trees by exploiting similarity information based on locality-sensitive hashing. We prove that our framework is secure without exposing the original record to other parties, while the computation overhead in the training process is kept low. Our experimental studies show that, compared with normal training with the local data of each party, our approach can significantly improve the predictive accuracy, and achieve comparable accuracy to the original GBDT with the data from all parties.

1 Introduction

Federated learning (FL) (McMahan et al. 2016; Mirhoseini, Sadeghi, and Koushanfar 2016; Shi et al. 2017; Yang et al. 2019; Mohri, Sivek, and Suresh 2019; Li et al. 2019a) has become a hot research area in machine learning. Federated learning addresses the privacy and security issues of model training in multiple parties. In reality, data are dispersed over different areas. For example, people tend to go to nearby hospitals, and the patient records in different hospitals are isolated. Ideally, hospitals may benefit more if they can collaborate with each other to train a model with the joint data. However, due to the increasing concerns and more regulations/policies on data privacy, organizations are not willing to share their own raw data records. Also, according to a recent survey (Yang et al. 2019), federated learning can be broadly categorized into *horizontal* federated learning, *vertical* federated learning and federated transfer learning. Much research efforts have been devoted to developing new learning algorithms in the setting of vertical or horizontal federated learning (Smith et al. 2017; Takabi, Hesamifard, and Ghasemi 2016; Liu, Chen, and Yang 2018; Yurochkin et al. 2019). We refer readers for more recent surveys for details (Yang et al. 2019; Li et al. 2019a).

On the other hand, Gradient Boosting Decision Trees (GBDTs) have become very successful in recent years by winning many awards in machine learning and data mining competitions (Chen and Guestrin 2016) as well as their effectiveness in many applications (Richardson, Dominowska, and Ragno 2007; Kim et al. 2009; Burges 2010; Li et al. 2019b). There have been several recent studies on how to train GBDTs in the federated learning setting (Cheng et al. 2019; Liu et al. 2019; Zhao et al. 2018). For example, SecureBoost (Cheng et al. 2019) developed vertical learning with GBDTs. In contrast, this study focuses on horizontal learning for GDBTs, where data samples with the same features are distributed among multiple parties.

There have been several studies of GDBT training in the setting of horizontal learning (Liu et al. 2019; Zhao et al. 2018). However, those approaches are not effective or efficient enough for practical use.

Model accuracy: The learned model may not have a good predictive accuracy. A recent study adopted differential privacy to aggregate distributed regression trees (Zhao et al. 2018). This approach boosts each tree only with the local data, which does not utilize the information of data from other parties. As we will show in the experiments, the model accuracy is much lower than our proposed approach.

Efficiency: The approach (Liu et al. 2019) has a prohibitively time-consuming learning process since it adopts complex cryptographic methods to encrypt the data from multiple parties. Due to a lot of extra cryptographic calculations, the approach brings prohibitively high overhead in the training process. Moreover, since GBDTs have to traverse the feature values to find the best split value, there is a huge number of comparison operations even in the building of a single node.

Considering the previous approaches' limitations on efficiency and model accuracy, this study utilizes a more practical privacy model as a tradeoff between privacy and efficiency/model accuracy (Du, Han, and Chen 2004; Liu, Chen, and Yang 2018). In this environment, a dishonest party might obtain some information about the other parties' data, but *it is still impossible for the dishonest party to derive the actual raw data of other parties*. Compared to differential privacy or secret sharing, this privacy model is weaker, but enables new opportunities for designing much more efficient and effective GBDTs.

Specifically, we propose a novel and practical federated learning framework for GDBTs (named SimFL). The basic idea is that instead of encryption on the feature values, we make use of the similarity between data of different parties in the training while protecting the raw data. First, we propose the use of locality-sensitive hashing (LSH) in the context of federated learning. We adopt LSH to collect similarity information without exposing the raw data. Second, we design a new approach called Weighted Gradient Boosting (WGB), which can build the decision trees by exploiting the similarity information with bounded errors. Our analysis show that SimFL satisfies the privacy model (Du, Han, and Chen 2004; Liu, Chen, and Yang 2018). The experimental results show that SimFL shows a good accuracy, while the training is fast for practical uses.

2 Preliminaries

Locality-Sensitive Hashing (LSH) LSH was first introduced by Gionis et al. (1999) for approximate nearest neighbor search. The main idea of LSH is to select a hashing function such that (1) the hash values of two neighbor points are equal with a high probability and (2) the hash values of two non-neighbor points are *not* equal with a high probability. A good property of LSH is that there are infinite input data for the same hash value. Thus, LSH has been used to protect user privacy in applications such as keyword searching (Wang et al. 2014) and recommendation systems (Qi et al. 2017).

The previous study (Datar et al. 2004) proposed the p-stable LSH family, which has been widely used. The hash function $\mathcal{F}_{\mathbf{a},b}$ is formulated as $\mathcal{F}_{\mathbf{a},b}(\mathbf{v}) = \lfloor \frac{\mathbf{a}\cdot\mathbf{v}+b}{r} \rfloor$, where $\mathbf{a}$ is a d-dimensional vector with entries chosen independently from a p-stable distribution (Zolotarev 1986); b is a real number chosen uniformly from the range $[0, r]$; r is a positive real number which represents the window size.

Gradient Boosting Decision Trees (GBDTs) The GBDT is an ensemble model which trains a sequence of decision trees. Formally, given a loss function l and a dataset with n instances and d features $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}(|\mathcal{D}| = n, \mathbf{x}_i \in \mathbb{R}^d, y_i \in \mathbb{R})$, GBDT minimizes the following objective function (Chen and Guestrin 2016).

$$\tilde{\mathcal{L}} = \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k) \tag{1}$$

where $\Omega(f) = \gamma T_l + \frac{1}{2}\lambda\|w\|^2$ is a regularization term to penalize the complexity of the model. Here γ and λ are hyper-parameters, T_l is the number of leaves and w is the leaf weight. Each f_k corresponds to a decision tree. Training the model in an additive manner, GBDT minimizes the following objective function at the t-th iteration.

$$\tilde{\mathcal{L}}^{(t)} = \sum_{i=1}^{n}[g_i f_t(\mathbf{x}_i) + \frac{1}{2}h_i f_t^2(\mathbf{x}_i)] + \Omega(f_t) \tag{2}$$

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 first and second order gradient statistics on the loss function. The decision tree is built from the root until reaching the restrictions such as the maximum depth. Assume I_L and I_R are the instance sets of left and right nodes after the split. Letting $I = I_L \cup I_R$, the gain of the split is given by

$$\mathcal{L}_{split} = \frac{1}{2}[\frac{(\sum_{i\in I_L} g_i)^2}{\sum_{i\in I_L} h_i + \lambda} + \frac{(\sum_{i\in I_R} g_i)^2}{\sum_{i\in I_R} h_i + \lambda} - \frac{(\sum_{i\in I} g_i)^2}{\sum_{i\in I} h_i + \lambda}] - \gamma \tag{3}$$

GBDT traverses all the feature values to find the split that maximizes the gain.

3 Problem Statement

This paper focuses on the application scenarios of horizontal federated learning. Multiple parties have their own data which have the same set of features. Due to data privacy requirements, they are not willing to share their private data with other parties. However, all parties want to exploit collaborations and benefits from a more accurate model that can be built from the joint data from all parties. Thus, *the necessary incentive for this collaboration is that federated learning should generate a much better learned model than the one generated from the local data of each party alone.* In other words, (much) better model accuracy is a pre-condition for such collaborations in horizontal federated learning. We can find such scenarios in various applications such as banks and healthcares (Yang et al. 2019).

Specifically, we assume that there are M parties, and each party is denoted by P_i ($i \in [1, M]$). We use $I_m = \{(\mathbf{x}_i^m, y_i^m)\}$ ($|I_m| = N_m, \mathbf{x}_i^m \in \mathbb{R}^d, y_i^m \in \mathbb{R}$) to denote the instance set of P_m. For simplicity, the instances have global IDs that are unique identifiers among parties (i.e., given two different instances $\mathbf{x}_i^m$ and $\mathbf{x}_j^n$, we have $i \neq j$).

Privacy model. The previous study (Du, Han, and Chen 2004) proposed a 2-party security model, which is also adopted in the previous studies (e.g., (Liu, Chen, and Yang 2018)). We extend the model for multiple parties, and we get the following privacy definition.

Definition 1 (Privacy Model)**.** Suppose all parties are *honest-but-curious*. For a protocol C performing $(O_1, O_2, ..., O_M) = C(I_1, I_2, ..., I_M)$, where $O_1, O_2, ..., O_M$ are parties $P_1, P_2, ..., P_M$'s output and $I_1, I_2, ..., I_M$ are their inputs, C is secure against P_1 if there exists infinite number of tuples $(I'_2, ..., I'_M, O'_2, ..., O'_M)$ such that $(O_1, O'_2, ..., O'_M) = C(I_1, I'_2, ..., I'_M)$.

Compared with the security definition in secure multiparty computation (Yao 1982), the privacy model in Definition 1 is weaker in the privacy level, as also discussed in the previous study (Du, Han, and Chen 2004). It does not handle the potential risks such as inference attacks. For example, it can happen that all the possible inputs for a certain output are close to each other, enough information about the input data may be disclosed (even though the exact information for the input is unknown). However, how likely those attacks can happen and its impact in real applications are still to be determined. Thus, like the previous studies (Du, Han, and Chen 2004; Liu, Chen, and Yang 2018), we view this model as a heuristics model for privacy protection. More importantly, with such a heuristic model, it is possible that we develop practical federated learning models that are much more efficient and have much higher accuracy than previous approaches.

Problem definition. The objective is to build an efficient and effective GBDT model under the privacy model in Definition 1 over the instance set $I = \bigcup_{i=1}^{M} I_i$ $(|I| = N)$.

4 The SimFL Framework

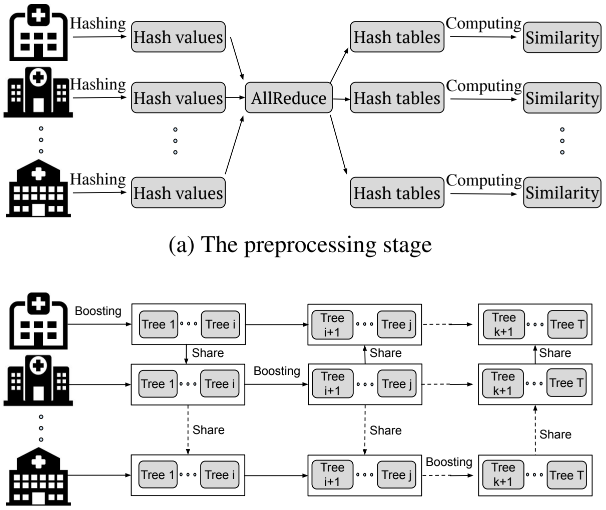
In this section, we introduce our framework, Similarity-based Federated Learning (SimFL), which enables the training of GBDTs in a horizontally federated setting.

An Overview of SimFL There are two main stages in SimFL: preprocessing and training. In practice, preprocessing can be done once and reuse for many runs of training. Only when the training data have updates, preprocessing has to be performed. Figure 1 shows the structures of these two stages. In the preprocessing stage, each party first computes the hash values using randomly generated LSH functions. Then, by collecting the hash values from LSH, multiple global hash tables are built and broadcast to all the parties, which can be modelled as an *AllReduce* communication operation (Patarasuk and Yuan 2009). Finally, each party can use the global hash tables for tree building without accessing other party's raw data. In the training stage, all parties together train a number of trees one by one using the similarity information. Once a tree is built in a party, it will be sent to all the other parties for the update of gradients. We obtain all the decision trees as the final learned model.

4.1 The Preprocessing Stage

For each instance, the aim of the preprocessing stage is to get the IDs of similar instances of the other parties. Specifically, for each party P_m, we want to get a matrix $\mathbf{S}^m \in \mathbb{R}^{N_m \times M}$, where $\mathbf{S}^m_{ij}$ is the ID of the instance in Party P_j that is similar with $\mathbf{x}^m_i$. To obtain the similarity of any two instances in the joint data without exposing the raw data to the other parties, we adopt the widely used p-stable LSH function (Datar et al. 2004). According to LSH, if two instances are similar, they have a higher probability to be hashed to the same value. Thus, by applying multiple LSH functions, the bigger the number of identical hash values of two instances, the more likely they are to be similar (Gionis et al. 1999).

Algorithm 1 shows the process of the preprocessing stage. Given L randomly generated p-stable hash functions, each party first computes the hash values of its instances. Then we build L global hash tables using an AllReduce operation, which has been well studied in the previous study (Patarasuk and Yuan 2009). Here the inputs to AllReduce are the instance IDs and their hash values of all parties. The reduction operation is to union the instances IDs with the same hash value. We adopt the bandwidth optimal and contention free design from the previous study (Patarasuk and Yuan 2009). After broadcasting the hash tables, each party can compute the similarity information. Specifically, in party P_m, given an instance $\mathbf{x}^m_i$, the similar instance in the other party P_j is the one with the highest count of identical hash values. If there are multiple instances with the same highest count, we randomly choose one as the similar instance. In this way, the data distribution of the other parties can be learned by adopting our weighted gradient boosting strategy, which we will show next.



(a) The preprocessing stage

(b) The training stage

Figure 1: An overview of the SimFL framework

4.2 The Training Stage

In the training stage, each party trains a number of trees sequentially. When party P_m is training a tree, to protect the individual records of other parties, only the local instances I_m are used to learn the tree. The learned trees are shared among the parties during the training process. To exploit the similarity information between the instances from different parties, we propose a new approach to build the decision tree, which is called Weighted Gradient Boosting (WGB). The basic idea is that an instance is important and representative if it is similar to many other instances. Since the gradients can represent the importance of an instance as shown in the previous studies (Chen and Guestrin 2016; Ke et al. 2017), we propose to use weighted gradients in the training. Next, we describe WGB in detail.

The Weighted Gradient Boosting Approach For an instance $\mathbf{x}^m_q \in I_m$, let $\mathbf{W}^n_{mq} = \{k|\mathbf{S}^n_{km} = q\}$, which contains the IDs of the instances in I_n that are similar with $\mathbf{x}^m_q$. Let g^m_q and h^m_q denote the first order and second order gradients of loss function at $\mathbf{x}^m_q$, respectively. When P_m is building a

Algorithm 1: The preprocessing stage

```
Input: LSH functions ${\mathcal{F}_k}_{k=1,2...L}$, Instance set $I$
Output: The similarity matrices $\mathbf{S}$
for each party $P_m$ do
    /* Conduct on party $P_m$ */
    for each instance $\mathbf{x}_i^m \in I_m$ do
        Compute hash values $\{\mathcal{F}_k(\mathbf{x}_i^m)\}_{k=1,2...L}$;
Building and broadcasting global hash tables by
  AllReduce;
for $m \leftarrow 1$ to $M$ do
    /* Conduct on party $P_m$ */
    for each instance $\mathbf{x}_i^m \in I_m$ do
        for $j = 1$ to $M$ do
            /* Collect similar instance IDs of $P_j$ */
            if $j \neq m$ then
                Find the instance ID $t$ with the highest
                  count of identical hash values;
                $\mathbf{S}_{ij}^m \leftarrow t$;
            else
                /* In the same party, the most similar
                  instance is itself */
                $\mathbf{S}_{ij}^m \leftarrow i$;
```

Algorithm 2: The process of learning a tree

```
Input: Instance set $I$
Output: A new decision tree
for $i = 1$ to $M$, $i \neq m$ do
    /* Conduct on party $P_i$ */
    $\mathbf{G}_{m*}^i \leftarrow \mathbf{0}$, $\mathbf{H}_{m*}^i \leftarrow \mathbf{0}$;
    Update the gradients of instances in $I_i$;
    for each instance $\mathbf{x}_q^i \in I_i$ do
        Get the similar instance ID $s = \mathbf{S}_{qm}^i$;
        $\mathbf{G}_{ms}^i \leftarrow \mathbf{G}_{ms}^i + g_q^i$, $\mathbf{H}_{ms}^i \leftarrow \mathbf{H}_{ms}^i + h_q^i$;
    Send $\mathbf{G}_{m*}^i$, $\mathbf{H}_{m*}^i$ to $P_m$;
/* Conduct on party $P_m$ */
Update the gradients of instances in $I_m$;
for each instance $\mathbf{x}_q^m \in I_m$ do
    $\mathbf{G}_{mq} \leftarrow 0$, $\mathbf{H}_{mq} \leftarrow 0$;
    for $i \leftarrow 1$ to $M$ do
        if $i == m$ then
            $\mathbf{G}_{mq} \leftarrow \mathbf{G}_{mq} + g_q^m$;
            $\mathbf{H}_{mq} \leftarrow \mathbf{H}_{mq} + h_q^m$;
        else
            $\mathbf{G}_{mq} \leftarrow \mathbf{G}_{mq} + \mathbf{G}_{mq}^i$;
            $\mathbf{H}_{mq} \leftarrow \mathbf{H}_{mq} + \mathbf{H}_{mq}^i$;
Build a tree with instances $I_m$ and weighted
  gradients $\mathbf{G}_{m*}$, $\mathbf{H}_{m*}$;
Send the tree to the other parties;
```

new tree at the t-th iteration, WGB minimizes the following objective function.

$$\tilde{\mathcal{L}}_w^{(t)} = \sum_{\mathbf{x}_q^m \in I_m} [\mathbf{G}_{mq} f_t(\mathbf{x}_q^m) + \frac{1}{2}\mathbf{H}_{mq} f_t^2(\mathbf{x}_q^m)] + \Omega(f_t)$$

$$\text{where } \mathbf{G}_{mq} = \sum_n \sum_{i \in \mathbf{W}_{mq}^n} g_i^n, \mathbf{H}_{mq} = \sum_n \sum_{i \in \mathbf{W}_{mq}^n} h_i^n \tag{4}$$

Compared with the objective in Eq. (2), Eq. (4) only uses the instances of I_m. Instead of using the gradients g_q^m, h_q^m of the instance $\mathbf{x}_q^m$, we use $\mathbf{G}_{mq}, \mathbf{H}_{mq}$ which are the sum of the gradients of the instances that are similar with $\mathbf{x}_q^m$ (including $\mathbf{x}_q^m$ itself). To help understanding, here is an example. Suppose we have two parties P_a and P_b. When computing the similarity information for a party P_a, the similar instance for both $\mathbf{x}_1^a$ and $\mathbf{x}_2^a$ may be $\mathbf{x}_3^b$. Then, when building trees in P_b, the gradient of $\mathbf{x}_3^b$ is replaced by the aggregated gradients of $\mathbf{x}_1^a$, $\mathbf{x}_2^a$, and $\mathbf{x}_3^b$. Considering $\mathbf{G}_{mq}, \mathbf{H}_{mq}$ as *weighted gradients*, we put more weights on instances that have a larger number of similar instances to utilize the similarity information.

Since the process of building a tree is similar between different parties, we only describe the process of building a tree in Party P_m, which is shown in Algorithm 2. At first, the parties update the gradients of the local instances. Then, for each instance of P_m, the other parties compute and send the aggregated gradients of the similar instances. Instead of sending each gradient directly, such aggregation on the local party can reduce the communication cost and protect the individual gradients. After all the aggregated gradients are computed and send to P_m, the weighted gradients can be easily computed by summing the aggregated gradients. Then, we can build a tree based on the weighted gradients.

5 Theoretical Analysis

5.1 Privacy Level Analysis

Theorem 1. The SimFL protocol satisfies the privacy model definition if $L < d$, where L is the number of hash functions and d is the number of dimensions of training data.

In short, there are infinite number of solutions if the number of unknowns (i.e., d) is bigger than the number of equations (i.e., L) (Ladyzhenskaia, Solonnikov, and Ural'ceva 1968). The detailed proof is available in Appendix A.

Theorem 1 indicates that, when $L < d$, SimFL ensures that, for any its output, there exists an infinite number of possible inputs resulting the same output. Therefore, a dishonest party cannot determine the actual raw data from other parties. Potentially, there may be background knowledge attack against SimFL. An optional method to resolve this problem is to decrease L to get a fewer number of equations, which results in a larger number of inputs for the same output. For example, if we know the background knowledge of a feature and now we only have $(d - 1)$ unknowns, we can set L to $(d-2)$ or even smaller to ensure that the raw data still cannot be extracted.

5.2 The Error of Weighted Gradient Boosting

Here we analyze the approximation error as well as the generalization error of WGB.

Theorem 2. For simplicity, we assume that the feature values of each dimension are i.i.d. uniform random variables and the split value is randomly chosen from the feature values. Suppose P_m is learning a new tree. We denote the approximation error of WGB as $\varepsilon^t = |\tilde{\mathcal{L}}_w^{(t)} - \tilde{\mathcal{L}}^{(t)}|$. Let $d_t = \max_{r,j} \|\mathbf{x}^m_{\mathbf{S}^j_{rm}} - \mathbf{x}^j_r\|_1$, $d_m = \max_{i,j} \|\mathbf{x}_i - \mathbf{x}_j\|_1$, $g' = \max_i |g_i|$, $h' = \max_i |h_i|$, and $f'_t = \max_i |f_t(\mathbf{x}_i)|$. Then, with probability at least $1 - \delta$, we have

$$\begin{aligned}\varepsilon^t \leq & \Big([1 - (1 - \frac{d_t}{d_m})^D](N - N_m) + \sqrt{\frac{(N - N_m)\ln\frac{1}{\delta}}{2}} \Big) \\ & \cdot (2g' f'_t + \frac{1}{2} h' f'^2_t)\end{aligned} \tag{5}$$

where D is the depth of the tree, N is the number of instances in I, and N_m is the number of instances in I_m.

The detailed proof of Theorem 2 is available in Appendix B. According to Theorem 2, the upper bound of the approximation error is $\mathcal{O}(N - N_m)$ with respect to the number of instances. The approximation error of WGB may increase as the number of instances of the other parties increases. However, let us consider the generalization error of WGB, which can be formulated as $\varepsilon^{WGB}_{gen}(t) = |\tilde{\mathcal{L}}_w^{(t)} - \tilde{\mathcal{L}}_*^{(t)}| \leq |\tilde{\mathcal{L}}_w^{(t)} - \tilde{\mathcal{L}}^{(t)}| + |\tilde{\mathcal{L}}^{(t)} - \tilde{\mathcal{L}}_*^{(t)}| \triangleq \varepsilon^t + \varepsilon_{gen}(t)$. The generalization error of vanilla GBDTs (i.e., $\varepsilon_{gen}(t)$) tends to decrease as the number of training instances N increases (Shalev-Shwartz and Ben-David 2014). Thus, WGB may still have a low generalization error as N increases.

5.3 Computation and Communication Efficiency

Here we analyze the computation and communication overhead of SimFL. Suppose we have T tress, M parties, N total training instances, and L hash functions.

Computation Overhead (1) In the preprocessing stage, we first need to compute the hash values, which costs $O(Nd)$. Then, we need to union the instance IDs with the same hash value and compute the similarity information. The union operation can be done in $\mathcal{O}(NL)$ since we have to traverse NL hash values in total. When computing the similarity information for an instance $\mathbf{x}^m_i$, a straightforward method is to union the instance IDs of L buckets with hash values $\{\mathcal{F}_k(\mathbf{x}^m_i)\}_{k=1,2...L}$ and conduct linear search to find the instance ID with the maximum frequency. On average, each hash bucket has a constant number of instances. Thus, for each instance, the linear search can be done in $\mathcal{O}(L)$. Thus, the total computation overhead in the preprocessing stage is $\mathcal{O}(NL + Nd)$ on average.

(2) In the training stage, the process of building a tree is the same as the vanilla GBDTs except computing the weighted gradients. The calculation of the weighted gradients is done by simple sum operations, which is $\mathcal{O}(N)$. Then, the computation overhead is $\mathcal{O}(NT)$ in the training.

Communication Overhead Suppose each real number uses 4 bytes to store. (1) In the preprocessing stage, according to the previous study (Patarasuk and Yuan 2009), since we have NL hash values and the corresponding instance IDs to share, the total communication cost in the AllReduce operation is $8MNL$ bytes. (2) In the training stage, each party has to send the aggregated gradients. The size of the gradients is no more than $8N$ (including g and h). After building a tree, P_m has to send the tree to the other parties. Suppose the depth of each tree is D. Our implementation uses 8 bytes to store the split value and the feature ID of each node. Therefore, the size of a tree is $8(2^D - 1)$. Since each tree has to be sent to $(M - 1)$ parties, the communication cost for one tree is $8(2^D - 1)(M - 1)$. The total communication overhead for a tree is $8[N + (2^D - 1)(M - 1)]$. Since there are T trees, the communication overhead in the training stage is $8T[N + (2^D - 1)(M - 1)]$.

6 Experiments

We present the effectiveness and efficiency of SimFL. To understand the model accuracy of SimFL, we compare SimFL with two approaches: 1) **SOLO**: Each party only trains vanilla GBDTs with its local data. This comparison shows the incentives of using SimFL. 2) **ALL-IN**: A party trains vanilla GBDTs with the joint data from all parties without the concern of privacy. This comparison demonstrates the potential accuracy loss of achieving the privacy model. We also compare SimFL with the distributed boosting framework proposed by Zhao et al. (2018) (referred as **TFL** (Tree-based Federated Learning)). Here we only adopt their framework and do not add any noise in the training (their system also adds noises to training for differential privacy).

We conducted the experiments on a machine running Linux with two Xeon E5-2640v4 10 core CPUs, 256GB main memory and a Tesla P100 GPU of 12GB memory. To take advantage of GPU, we use ThunderGBM (Wen et al. 2019) in our study. We use six public datasets from the LIBSVM website[1], as listed in Table 1. We use 75% of the datasets for training and the remainder for testing. The maximum depth of the trees is set to 8. For the LSH functions, we choose $r = 4.0$ and $L = \min\{40, d - 1\}$, where d is the dimension of the dataset. The total number of trees is set to 500 in all approaches. Due to the randomness of the LSH functions, the results of SimFL may differ in the different runnings. We run SimFL for 10 times in each experiment and report the average, minimal and maximum errors.

In reality, the distribution of the data in different parties may vary. For example, due to the ozone hole, the countries in the Southern Hemisphere may have more skin cancer patients than the Northern Hemisphere. Thus, like a previous work (Yurochkin et al. 2019), we consider two ways to divide the training dataset to simulate different data distributions in the federated setting: *unbalanced partition* and *balanced partition*. In the unbalanced partition, we divide the datasets with two classes (i.e., 0 and 1) to subsets where each subset has a relatively large proportion of the instances

[1] https://www.csie.ntu.edu.tw/~cjlin/libsvm/index.html

Table 1: datasets used in the experiments

dataset	cardinality	dimension	data size
a9a	32,561	123	16MB
cod-rna	59,535	9	2.1MB
real-sim	72,309	20,958	6.1GB
ijcnn1	49,990	22	4.4MB
SUSY	1,000,000	18	72MB
HIGGS	1,000,000	28	112MB

Table 2: The test errors of different approaches ($\theta = 80\%$)

datasets	SimFL			TFL	SOLO$_A$	SOLO$_B$	ALL-IN
	avg	min	max				
a9a	17.0%	16.7%	17.2%	23.1%	19.1%	22.0%	15.1%
cod-rna	6.5%	6.3%	6.7%	7.5%	9.6%	8.2%	6.13%
real-sim	8.3%	8.2%	8.4%	10.1%	11.8%	14.4%	6.5%
ijcnn1	4.5%	4.5%	4.6%	4.7%	5.9%	4.9%	3.7%
SUSY	23.3%	23.1 %	23.4%	32.6%	25.4%	32.8 %	21.38%
HIGGS	30.9%	30.8%	31.0%	36.1%	37.1%	35.6%	29.4%

in one class. Given a ratio $\theta \in (0, 1)$, we randomly sample θ proportion of the instances that are with the class 0 and $(1-\theta)$ proportion of the instances that are with the class 1 to form a subset, and the remained for the other subset. Then, we can split these two subsets to more parts equally and randomly to simulate more parties. In the balanced partition, we simply split the datasets equally and randomly assign the instances to different parties.

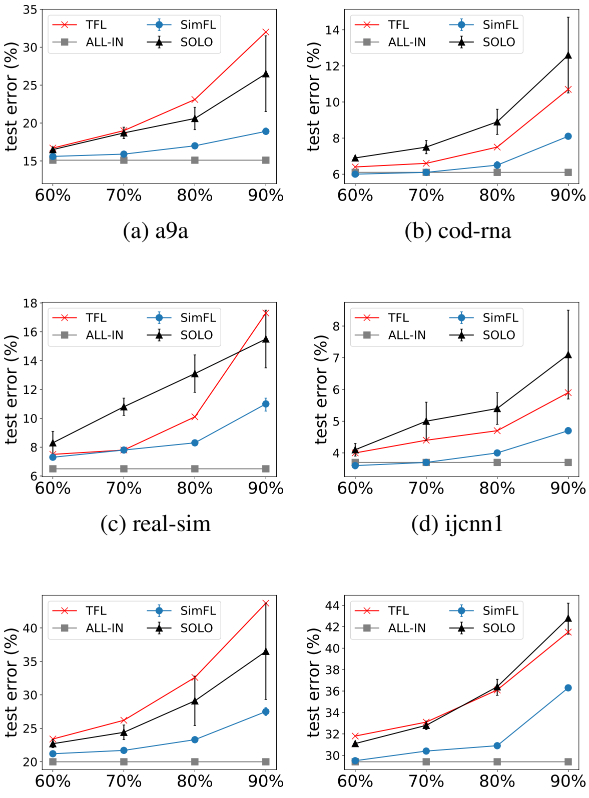
6.1 Test Errors

We first divide the datasets into two parts using the unbalanced partition with the ratio $\theta = 80\%$, and assign them to two parties A and B . The test errors are shown in Table 2. We have the following observations. First, the test errors of SimFL are always lower than SOLO on data parts A and B (denoted as SOLO$_A$ and SOLO$_B$ respectively). The accuracy can be improved by about 4% on average by performing SimFL. Second, the test error of SimFL is close to ALL-IN. Third, Compared with TFL, SimFL has much lower test errors. The test errors of TFL are always bigger than SOLO, which discourages the adoptions of TFL in practice. In other words, TFL's approach on aggregating decision trees from multiple parties cannot improve the prediction of the local data of individual parties.

To figure out the impact of the ratio θ, we conduct experiments with different ratios that range from 60% to 90%. The results are shown in Figure 2. Compared with TFL and SOLO, SimFL works quite well especially when θ is high. The similarity information can effectively help to discover the distribution of the joint data among multiple parties. The variation of SimFL is low in multiple runs.

Table 3: The test errors of different approaches (balanced partition)

datasets	SimFL			TFL	SOLO$_A$	SOLO$_B$	ALL-IN
	avg	min	max				
a9a	15.1%	14.9%	15.2%	15.6%	15.5%	15.9%	15.1%
cod-rna	6.0%	5.9%	6.1%	6.3%	6.4%	6.7%	6.1%
real-sim	7.1%	6.9%	7.2%	7.4%	8.4%	7.9%	6.5%
ijcnn1	3.6%	3.5%	3.7%	3.8%	3.8%	4.2%	3.7%
SUSY	19.6%	19.4%	19.9%	20.2%	20.1%	20.1%	20.0%
HIGGS	29.3%	29.2%	29.5%	31.4%	30.2%	30.5%	29.3%



(a) a9a

(b) cod-rna

(c) real-sim

(d) ijcnn1

(e) SUSY

(f) HIGGS

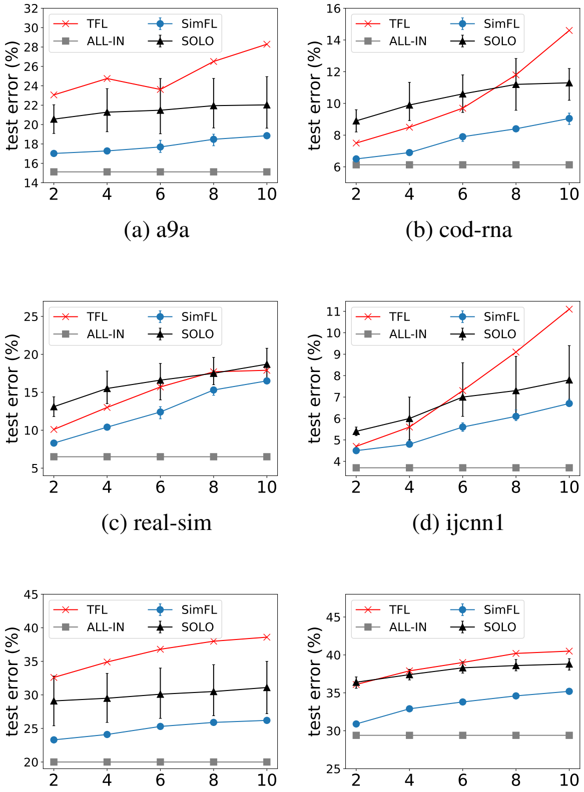
Figure 2: The test error with different ratio θ

Table 3 shows the errors with the setting of the balanced partition. It seems that for the six datasets, the local data in each party is good enough to train the model. The test errors of SOLO and ALL-IN are very close to each other. However, our SimFL still performs better than SOLO and TFL, and sometimes even has a lower test error than ALL-IN. Even in a balanced partition, SimFL still has a good performance.

Figure 3 and Figure 4 show the test errors with different number of parties. The ratio θ is set to 80% for the unbalanced partition. As the number of parties increases, according to Section 5.2, the upper bound of the generalization error of SimFL may increase since N is fixed and $(N - N_m)$ increases, which agrees with the experimental results. Still, SimFL has a lower test error than the minimal error of SOLO at most times. While the test error of TFL changes dramatically as the number of parties increases, SimFL is much more stable in both balanced and unbalanced data partition.

6.2 Efficiency

To show the training time efficiency of SimFL, we present the time and communication cost in the training stage and the preprocessing stage (denoted as *prep*) in Table 4. The number of parties is set to 10 and we adopt a balanced par-



(a) a9a (b) cod-rna

(c) real-sim (d) ijcnn1

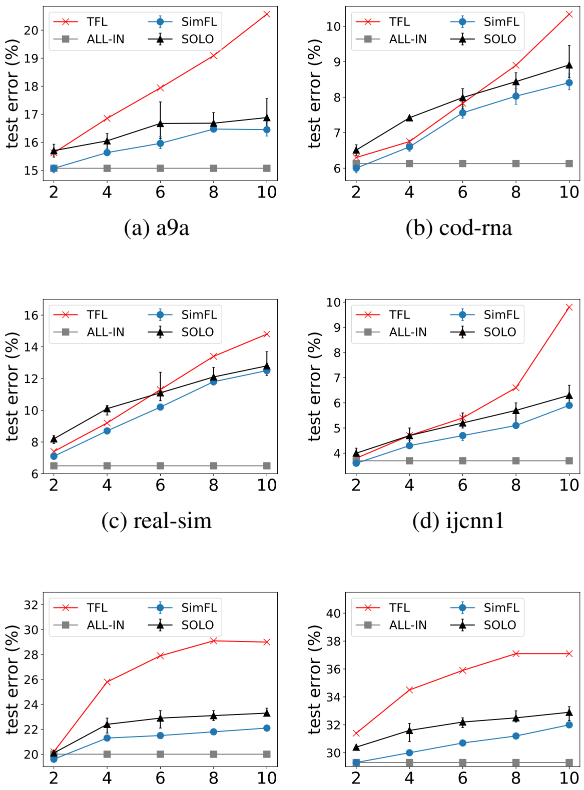
(e) SUSY (f) HIGGS

Figure 3: The impact of number of parties ($\theta = 80\%$)

tition here. First, we can observe that the training time of SimFL is very close to SOLO. The computation overhead is less than 10% of the total training time. Second, since SimFL only needs the local instances to build a tree while ALL-IN needs all the instances, the training time of SimFL is smaller than ALL-IN. Moreover, the proprecessing overhead is acceptable since it may only be required to perform once and reused for future training. One common scenario is that a user may try many runs of training with different hyper parameter settings and the pre-processing cost can be amortized among those runs. Last, the communication overhead per tree is very low, which costs no more than 10MB. In the encryption methods (Liu et al. 2019), since additional keys need to be transferred, their communicated data size per tree can be much larger than ours.

7 Conclusions

The success of federated learning highly relies on the training time efficiency and the accuracy of the learned model. However, we find that existing horizontal federated learning systems of GBDTs suffer from low efficiency and/or low model accuracy. Based on an established relaxed privacy model, we propose a practical federated learning framework **SimFL** for GBDTs by exploiting similarity. We take advantage of efficient locality-sensitive hashing to collect the similarity information without exposing the individual records, in contrast of costly secret sharing/encryption operations in previous studies. By designing a weighted gradient boosting method, we can utilize the similarity information to build decision trees with bounded errors. We prove that SimFL satisfies the privacy model. The experiments show that SimFL significantly improves the predictive accuracy compared with training with data in individual parties alone, and is close to the model with joint data from all parties.



(a) a9a (b) cod-rna

(c) real-sim (d) ijcnn1

(e) SUSY (f) HIGGS

Figure 4: The impact of number of parties (balanced partition)

Table 4: The training time (sec), preprocessing time (sec), communication cost (MB) per party in the preprocessing and communication cost (MB) per tree in the training.

datasets	ALL-IN	SOLO	SimFL			
	training time	training time	time		communication	
			training	prep	training	prep
a9a	21.6	15.2	17.2	135	0.28	10.4
cod-rna	24.7	16.3	17.9	189	0.49	4.3
real-sim	69.5	32.4	34.5	380	0.6	23.1
ijcnn1	29.3	15.7	17.4	228	0.42	8.4
SUSY	204.1	34.6	43.1	963	8.0	136
HIGGS	226.6	39.8	44.8	996	8.0	216

Acknowledgements

This work is supported by a MoE AcRF Tier 1 grant (T1 251RES1824) and a MOE Tier 2 grant (MOE2017-T2-1-122) in Singapore. The authors thank Qiang Yang and Yang Liu for their insightful suggestions.

References

Burges, C. J. 2010. From ranknet to lambdarank to lambdamart: An overview. *Learning* 11(23-581):81.

Chen, T., and Guestrin, C. 2016. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 785–794. ACM.

Cheng, K.; Fan, T.; Jin, Y.; Liu, Y.; Chen, T.; and Yang, Q. 2019. Secureboost: A lossless federated learning framework. *arXiv preprint arXiv:1901.08755*.

Datar, M.; Immorlica, N.; Indyk, P.; and Mirrokni, V. S. 2004. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*, 253–262. ACM.

Du, W.; Han, Y. S.; and Chen, S. 2004. Privacy-preserving multivariate statistical analysis: Linear regression and classification. In *SDM*, 222–233. SIAM.

Gionis, A.; Indyk, P.; Motwani, R.; et al. 1999. Similarity search in high dimensions via hashing. In *Vldb*, volume 99, 518–529.

Ke, G.; Meng, Q.; Finley, T.; Wang, T.; Chen, W.; Ma, W.; Ye, Q.; and Liu, T.-Y. 2017. Lightgbm: A highly efficient gradient boosting decision tree. In *NeurIPS*.

Kim, S.-M.; Pantel, P.; Duan, L.; and Gaffney, S. 2009. Improving web page classification by label-propagation over click graphs. In *Proceedings of the 18th ACM conference on Information and knowledge management*, 1077–1086. ACM.

Ladyzhenskaia, O. A.; Solonnikov, V. A.; and Ural'ceva, N. N. 1968. *Linear and quasi-linear equations of parabolic type*, volume 23. American Mathematical Soc.

Li, Q.; Wen, Z.; Wu, Z.; Hu, S.; Wang, N.; and He, B. 2019a. Federated learning systems: Vision, hype and reality for data privacy and protection. *arXiv preprint arXiv:1907.09693*.

Li, Q.; Wu, Z.; Wen, Z.; and He, B. 2019b. Privacy-preserving gradient boosting decision trees. *AAAI-20, arXiv preprint arXiv:1911.04209*.

Liu, Y.; Ma, Z.; Liu, X.; Ma, S.; Nepal, S.; and Deng, R. 2019. Boosting privately: Privacy-preserving federated extreme boosting for mobile crowdsensing. *arXiv preprint arXiv:1907.10218*.

Liu, Y.; Chen, T.; and Yang, Q. 2018. Secure federated transfer learning. *arXiv preprint arXiv:1812.03337*.

McMahan, H. B.; Moore, E.; Ramage, D.; and y Arcas, B. A. 2016. Federated learning of deep networks using model averaging. *CoRR* abs/1602.05629.

Mirhoseini, A.; Sadeghi, A.-R.; and Koushanfar, F. 2016. Cryptoml: Secure outsourcing of big data machine learning applications. In *2016 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*.

Mohri, M.; Sivek, G.; and Suresh, A. T. 2019. Agnostic federated learning. In Chaudhuri, K., and Salakhutdinov, R., eds., *ICML*, volume 97 of *Proceedings of Machine Learning Research*, 4615–4625. PMLR.

Patarasuk, P., and Yuan, X. 2009. Bandwidth optimal all-reduce algorithms for clusters of workstations. *Journal of Parallel and Distributed Computing* 69(2):117–124.

Qi, L.; Zhang, X.; Dou, W.; and Ni, Q. 2017. A distributed locality-sensitive hashing-based approach for cloud service recommendation from multi-source data. *IEEE Journal on Selected Areas in Communications* 35(11):2616–2624.

Richardson, M.; Dominowska, E.; and Ragno, R. 2007. Predicting clicks: estimating the click-through rate for new ads. In *Proceedings of the 16th international conference on World Wide Web*, 521–530. ACM.

Shalev-Shwartz, S., and Ben-David, S. 2014. *Understanding machine learning: From theory to algorithms*. Cambridge university press.

Shi, E.; Chan, T.-H. H.; Rieffel, E.; and Song, D. 2017. Distributed private data analysis: Lower bounds and practical constructions. *ACM Transactions on Algorithms (TALG)* 13(4):50.

Smith, V.; Chiang, C.-K.; Sanjabi, M.; and Talwalkar, A. S. 2017. Federated multi-task learning. In *Advances in Neural Information Processing Systems*, 4424–4434.

Takabi, H.; Hesamifard, E.; and Ghasemi, M. 2016. Privacy preserving multi-party machine learning with homomorphic encryption. In *29th Annual Conference on Neural Information Processing Systems*.

Wang, B.; Yu, S.; Lou, W.; and Hou, Y. T. 2014. Privacy-preserving multi-keyword fuzzy search over encrypted data in the cloud. In *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*, 2112–2120. IEEE.

Wen, Z.; Shi, J.; Li, Q.; He, B.; and Chen, J. 2019. Thundergbm: Fast gbdts and random forests on gpus. In *https://github.com/Xtra-Computing/thundergbm*.

Yang, Q.; Liu, Y.; Chen, T.; and Tong, Y. 2019. Federated machine learning: Concept and applications. *ACM Trans. Intell. Syst. Technol.* 10(2):12:1–12:19.

Yao, A. C.-C. 1982. Protocols for secure computations. In *FOCS*, volume 82, 160–164.

Yurochkin, M.; Agarwal, M.; Ghosh, S.; Greenewald, K.; Hoang, N.; and Khazaeni, Y. 2019. Bayesian nonparametric federated learning of neural networks. In Chaudhuri, K., and Salakhutdinov, R., eds., *ICML*, volume 97 of *Proceedings of Machine Learning Research*, 7252–7261. PMLR.

Zhao, L.; Ni, L.; Hu, S.; Chen, Y.; Zhou, P.; Xiao, F.; and Wu, L. 2018. Inprivate digging: Enabling tree-based distributed data mining with differential privacy. In *INFOCOM*, 2087–2095. IEEE.

Zolotarev, V. M. 1986. *One-dimensional stable distributions*, volume 65. American Mathematical Soc.

A Proof of Theorem 1

Proof. Without loss of generality, we only need to prove that the protocol is secure against P_m. In the whole FL process, P_m knows the hash values of the other instances and the aggregated gradients of the similar instances. Next, we prove that there are infinite number of instances that can provide the same hash values and gradients.

Given the hash values, P_m knows L different compound inequalities about an instance $\mathbf{x}_k$ with the form $\mathcal{F}_j(\mathbf{x}_k) \leq \frac{\mathbf{a}_j \cdot \mathbf{x}_k + b_j}{r} < \mathcal{F}_j(\mathbf{x}_k) + 1$, $j = 1..L$. Consider a stricter system of L different linear equations $\frac{\mathbf{a}_j \cdot \mathbf{x}_k + b_j}{r} = z_j$, where $z_j \in [\mathcal{F}_j(\mathbf{x}_k), \mathcal{F}_j(\mathbf{x}_k) + 1)$. Obviously, for a linear system, there are no solution or an infinite number of solutions if the number of unknowns (i.e., d) is bigger than the number of equations (i.e., L) (Ladyzhenskaia, Solonnikov, and Ural'ceva 1968). Since we already know there is at least one solution (i.e., $\mathbf{x}_k$), the number of solutions is infinite. Thus, there are infinite number of instances that can result in the same hash values that P_m knows. Since the gradients are computed based on the prediction values, the instances can provide the same gradient as long as they have the same prediction values in the trees. Note that the restrictions in the decision trees only specify a range of the feature values. So we have infinite number of instances that can provide the same gradients as long as they satisfy the same restrictions on the features values and are divided into the same leaf. Thus, we can have infinite number of instances that can provide the same output for party P_m. □

B Proof of Theorem 2

Proof. For ease of presentation, we suppose the instances have different globally IDs (i.e., for any two instances $\mathbf{x}_i^m$ and $\mathbf{x}_j^n$, if $m \neq n$ then $i \neq j$). Thus, we can use g_i instead of g_i^m to denote the first order gradient of an instance $\mathbf{x}_i^m$. Also, we use h_i instead of h_i^m to denote the second order gradient of $\mathbf{x}_i^m$. We have

$$\begin{aligned}
\tilde{\mathcal{L}}^{(t)} &= \sum_{i=1}^{n} [g_i f_t(\mathbf{x}_i) + \frac{1}{2} h_i f_t^2(\mathbf{x}_i)] + \Omega(f_t) \\
&= \sum_{\mathbf{x}_q^m \in I_m} [g_q f_t(\mathbf{x}_q^m) + \frac{1}{2} h_q f_t^2(\mathbf{x}_q^m)] \\
&+ \sum_{\substack{j=1 \\ j \neq m}}^{M} \sum_{\mathbf{x}_r^j \in I_j} [g_r f_t(\mathbf{x}_r^j) + \frac{1}{2} h_r f_t^2(\mathbf{x}_r^j)] + \Omega(f_t)
\end{aligned} \tag{6}$$

With Eq. (4), we have

$$\begin{aligned}
\tilde{\mathcal{L}}^{(t)} &= \tilde{\mathcal{L}}_w^{(t)} - \sum_{\substack{j=1 \\ j \neq m}}^{M} \sum_{\substack{r \\ \mathbf{x}_r^j \in I_j}} [g_r f_t(\mathbf{x}^m_{\mathbf{S}^j_{rm}}) + \frac{1}{2} h_r f_t^2(\mathbf{x}^m_{\mathbf{S}^j_{rm}})] \\
&+ \sum_{\substack{j=1 \\ j \neq m}}^{M} \sum_{\mathbf{x}_r^j \in I_j} [g_r f_t(\mathbf{x}_r^j) + \frac{1}{2} h_r f_t^2(\mathbf{x}_r^j)] \\
&= \tilde{\mathcal{L}}_w^{(t)} - \sum_{\substack{j=1 \\ j \neq m}}^{M} \sum_{\substack{r \\ \mathbf{x}_r^j \in I_j}} [g_r (f_t(\mathbf{x}^m_{\mathbf{S}^j_{rm}}) - f_t(\mathbf{x}_r^j)) \\
&+ \frac{1}{2} h_r (f_t^2(\mathbf{x}^m_{\mathbf{S}^j_{rm}}) - f_t^2(\mathbf{x}_r^j))]
\end{aligned} \tag{7}$$

Thus, we have

$$\begin{aligned}
\varepsilon^t &= |\tilde{\mathcal{L}}_w^{(t)} - \tilde{\mathcal{L}}^{(t)}| \\
&= \sum_{\substack{j=1 \\ j \neq m}}^{M} \sum_{\substack{r \\ \mathbf{x}_r^j \in I_j}} |g_r (f_t(\mathbf{x}^m_{\mathbf{S}^j_{rm}}) - f_t(\mathbf{x}_r^j)) \\
&+ \frac{1}{2} h_r (f_t^2(\mathbf{x}^m_{\mathbf{S}^j_{rm}}) - f_t^2(\mathbf{x}_r^j))|
\end{aligned} \tag{8}$$

Let $\xi_r = |g_r(f_t(\mathbf{x}^m_{\mathbf{S}^j_{rm}}) - f_t(\mathbf{x}_r^j)) + \frac{1}{2} h_r (f_t^2(\mathbf{x}^m_{\mathbf{S}^j_{rm}}) - f_t^2(\mathbf{x}_r^j))|$. Since $g' = \max_i |g_i|$, $h' = \max_i |h_i|$, and $f_t' = \max_i |f_t(\mathbf{x}_i)|$, we have

$$\xi_r \leq 2 g' f_t' + \frac{1}{2} h' f_t'^2 \tag{9}$$

Notice that $\xi_r = 0$ if $\mathbf{x}^m_{\mathbf{S}^j_{rm}}$ and $\mathbf{x}_r^j$ go to the same leaf of the tree f_t. According to our assumption, the feature values of each feature are i.i.d. uniform random variables and the split value is randomly chosen from the feature values. Let $d_t = \max_{r,j} \|\mathbf{x}^m_{\mathbf{S}^j_{rm}} - \mathbf{x}_r^j\|_1$ and $d_m = \max_{i,j} \|\mathbf{x}_i - \mathbf{x}_j\|_1$, then the probability that $\mathbf{x}^m_{\mathbf{S}^j_{rm}}$ and $\mathbf{x}_r^j$ are divided into two directions in each node is smaller than $\frac{d_t}{d_m}$ (i.e., the probability that randomly drop a ball in a line with length d_m and it falls in a interval with length d_t). Let D denotes the depth of the tree. Then, the probability that $\mathbf{x}^m_{\mathbf{S}^j_{rm}}$ and $\mathbf{x}_r^j$ go to the same leaf is bigger than $(1 - \frac{d_t}{d_m})^D$. There are $(N - N_m)$ different ξ_r. Let H denotes the nubmer of times that $\xi_r \neq 0$. By Hoeffding's inequality with a Bernoulli distribution, with probability at least $1 - \delta$, we have

$$H \leq [1 - (1 - \frac{d_t}{d_m})^D](N - N_m) + \sqrt{\frac{(N - N_m) \ln \frac{1}{\delta}}{2}} \tag{10}$$

With Eq. (9) and Eq. (10), we have

$$\begin{aligned}
\varepsilon^t \leq & \Big([1 - (1 - \frac{d_t}{d_m})^D](N - N_m) + \sqrt{\frac{(N - N_m) \ln \frac{1}{\delta}}{2}} \Big) \\
& \cdot (2 g' f_t' + \frac{1}{2} h' f_t'^2)
\end{aligned} \tag{11}$$

□