Efficient Distributed Decision Trees for Robust Regression [Technical Report]

Tian Guo[1], Konstantin Kutzkov[2], Mohamed Ahmed[2], Jean-Paul Calbimonte[1], and Karl Aberer[1]

[1] Ećole Polytechnique Fédérale de Lausanne (EPFL)
{tian.guo,jean-paul.calbimonte,karl.aberer}@epfl.ch,
[2] NEC Laboratories, Europe
{mohamed.ahmed}@neclab.eu
{kutzkov}@gmail.com

Abstract. The availability of massive volumes of data and recent advances in data collection and processing platforms have motivated the development of distributed machine learning algorithms. In numerous real-world applications large datasets are inevitably noisy and contain outliers. These outliers can dramatically degrade the performance of standard machine learning approaches such as regression trees. To this end, we present a novel distributed regression tree approach that utilizes robust regression statistics, statistics that are more robust to outliers, for handling large and noisy data. We propose to integrate robust statistics based error criteria into the regression tree. A data summarization method is developed and used to improve the efficiency of learning regression trees in the distributed setting. We implemented the proposed approach and baselines based on Apache Spark, a popular distributed data processing platform. Extensive experiments on both synthetic and real datasets verify the effectiveness and efficiency of our approach.

Keywords: Decision Tree, Distributed Machine Learning, Robust Regression, Data Summarization

1 Introduction

Decision trees are at the core of several highly successful machine learning models for both regression and classification, since their introduction by Quinlan [19]. Their popularity stems from the ability to (a) select, from the set of all attributes, a subset that is most relevant for the regression and classification problem at hand; (b) identify complex, non-linear correlations between attributes; and to (c) provide highly interpretable and human-readable models [7, 17, 19, 25]. Recently due to the increasing amount of available data and the ubiquity of distributed computation platforms and clouds, there is a rapidly growing interest in designing distributed versions of regression and classification trees [1,2,17,21,26,28], for instance, the decision/regression tree in Apache Spark MLlib machine learning package[3]. Meanwhile, since many of the large

[3] `http://spark.apache.org/docs/latest/mllib-decision-tree.html`

datasets are from observations and measurements of physical entities and events, such data is inevitably noisy and skewed in part due to equipment malfunctions or abnormal events [10,12,27].

With this paper, we propose an efficient distributed and regression tree learning framework that is robust to noisy data with outliers. This is a significant contribution since the effect of outliers on conventional regression trees based on the mean squared error criterion is often disastrous. Noisy datasets contain outliers (e.g., grossly mismeasured target values), which deviate from the distribution followed by the bulk of the data. Ordinary (distributed) regression tree learning minimizes the squared mean error objective function and outputs the mean of the data points in the leaf nodes as predictions, which is especially problematic and sensitive to noisy data in two aspects [10,12,25]. First, during the tree growing phase (the learning phase), internal tree nodes are split so as to minimize the square-error loss function, which places much more emphasis on observations with large residuals [7,10,25]. As a result, bias on the split of a tree node due to noisy and skewed data will propagate to descendent nodes and derail the tree building process. Second, outliers drag the mean predictions away from the true values on leaf nodes, thereby leading to highly skewed predictors. Consequentially, the distributed regression tree trained on noisy data can neither identify the true patterns in data, nor provide reliable predictions [8,10,12,25,27].

Previous methods to address robustness in the distributed regression tree fail to prevent noisy data from deviating the splits and predictions of tree nodes. For regression problems, it can be very difficult to spot noise or outliers in the data without careful investigation and even harder in multivariate data sets with both categorical and numerical features [13]. Overfitting avoidance, known as the node pruning in the context of regression trees, is a general way to allow robustness for unseen data by penalizing the tree for being too complex. But pruning operations cannot correct the biased splits of tree nodes [10,12]. Ensemble methods like RandomForest [15], RotationForest [20] and Gradient Boosted Tree [7] produce superior results by creating a large number of trees. But outliers distributed across attributes (or features) would still bias individual trees as well as the predictions aggregated from them.

Contributions: In this paper, we focus on enhancing the robustness of a distributed regression tree as well as the training efficiency. Concretely, this paper makes the following contributions:

- We define the distributed robust regression tree employing robust loss functions and identify the difficulty in designing an efficient training algorithm for the distributed robust regression tree.
- We propose a novel distributed training framework for the robust regression tree, which consists an efficient data summarization method on distributed data and a tree growing approach exploiting the data summarization to evaluate robust loss functions.
- The proposed distributed robust regression tree and baselines are implemented based on Apache Spark. Extensive experiments on both synthetic and real datasets demonstrate the efficiency and effectiveness of our approach.

The organization of the paper is as follows: Section 2 summarizes the related work. Section 3 presents the necessary background and the problem definition. Then, Section 4 and Section 5 present proposed framework and experiment results. We discuss the possible extension in Section 6 and conclude the work in Section 7.

2 Related Work

To the best of our knowledge, there is no existing work on the robust loss function based distributed regression trees in the literature, and thus we first summarize previous efforts to handle noisy data for regression/classification trees in centralized environments, and then the work on the distributed regression/classification trees. and the data summarization techniques utilized in the distributed regression trees.

Robust classification/regression trees: Many methods have been proposed to handle noisy data, but most of them concentrate on refining leaf nodes after training or purely on the classification problem. [29] applies smoothing on the leaves of a decision tree but not inner nodes. [5] assigns a confidence score to the classifier predictions rather than improving the classification itself. Zadorny and Elkan [29], Provost and Domingos [18] and [3] improve the classification probabilities by using regression in the leaves. Another well-known method for dealing with noisy data is fuzzy decision trees [8,16]. The fuzzy function may be domain specific and require a human expert in order to correctly define it. The other type of approaches is based on post-processing applied after a decision tree has already been built on noisy data. John [10] proposed iterative removal of instances with outlier values. [12] requires to perform back-ward path traversal for examined instances.

Our paper aims to improve the robustness of distributed regression trees by preventing the outliers from influencing the tree induction phase based on robust loss functions. Above post-processing methods can be smoothly integrated into our framework.

Distributed classification/regression trees: Our proposed approach borrows ideas from previous distributed regression tree algorithms to improve the training efficiency. But the previous algorithms do not consider the effect of data noise and outlier issues.

Parallel and distributed decision tree algorithms can be grouped into two main categories: task-parallelism and data-parallelism. Algorithms in the first category [4, 23] divide the tree into sub-trees, which are constructed on different workers, e.g. after the first node is split, the two remaining sub-trees are constructed on separate workers. The downside of this approach is that each worker should either have a full copy of data. For large data sets, this method would lead to slowdown rather than speed-up.

In the data-parallelism approach, the training instances are divided among the different nodes of the cluster. Dividing data by features [6] requires the workers to coordinate which input data instance falls into which tree-node. This requires additional communication, which we try to avoid as we scale to very large data sets. Dividing the data by instances [21] avoids this problem. Instance-partitioning approach PLANET [17] selects splits using histograms with fixed bins constructed over the value domain of features. Such static histograms overlooks the variation of underlying data distribution as the tree grows and therefore could lead to biased splits. [2, 26] put forward to construct dynamic histograms rebuilt for each layer of tree nodes and used for deliberately approximating the exact splits. [2, 26] communicate the histograms re-built for each layer of tree nodes to a master worker for tree induction. [1] is a MapReduce algorithm which builds multiple random forest ensembles on distributed blocks of data and merges them into a mega-ensemble. In [11] ScalParC employs a distributed hash table to implement the splitting phase for classification problems. [9] approach uses sampling to achieve memory efficient processing of numerical attributes for Gini impurity in the classification tree.

In this paper, our approach falls into the instance-partition category and we build dynamic histograms to summarize the value distribution of the target variable for the robust loss estimation.

Data summarization in distributed classification/regression trees: Data summarization in distributed regression trees [2,17,22,26] serves for data compression to facilitate the communication between workers and the master and supports mergeable operations for building a global picture about the data distribution on the master to grow the tree. Meanwhile, [2, 17, 26] build histograms over the feature value domain to provide splits candidates in growing the tree. Our proposed data summarization borrows ideas from [2] and is able to support efficient estimation of robust loss criteria.

3 Preliminaries and Problem Statement

In this part, we first present the regression tree employing robust loss functions. Then, we describe the robust regression tree in the distributed environment and formulate the problem of this paper.

3.1 Robust Regression Tree

In the regression problem, define a dataset $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}$, where $\boldsymbol{x}_i \in \mathbb{N}^d$ is a vector of predictor features of a data instance and $y_i \in \mathbb{R}$ is the target variable. d is the number of features. Let $D^n \in \mathcal{D}$ denote the set of instances falling under tree node n.

Regression tree construction [19,25] proceeds by repeated greedy expansion of tree nodes layer by layer until a stopping criterion, e.g. the tree depth is met. Initially, all data instances belong to the root node of the tree. An internal tree node (e.g., D^n) is split into two children nodes respectively with data subsets $D_L (D_L \subset D^n)$ and $D_R (D_R = D^n - D_L)$ by using a predicate on a feature, so as to minimize the weighted loss criteria: $\frac{|D_L|}{|D^n|} L(D_L) + \frac{|D_R|}{|D^n|} L(D_R)$, where $L(\cdot)$ is a loss function (or error criteria) defined over a set of data instances.

This paper proposes the distributed regression tree employing robust loss functions to handle noisy datasets with outliers on the target variable (the regression tree is robust to outliers in feature space [7]). In robust regression, there are two main types of robust loss functions: accommodation and rejection [7, 10, 24]. Accommodation approach is to define a loss function that lessens the impact of outliers The least absolute deviation, referred to as LAD, is an accommodation method [7, 24, 25]. It is defined on a set of data instances D as: $L_l(D) = \frac{1}{|D|} \sum_{(\boldsymbol{x}_i y_i) \in D} |y_i - \hat{y}|$, and $\hat{y} = median_{(\boldsymbol{x}_i y_i) \in D}(\{y_i\})$, which returns the median of a set of values [25]. On the other hand, rejection approach aims to restrict the attention only to the data that seems ”normal” [10]. The loss function of the rejection type is the trimmed least absolute deviation, referred to as TLAD. It is defined as $L_l(\tilde{D})$, where $\tilde{D}$ is the trimmed dataset of D derived by removing data instances with the $k\%$ largest and $k\%$ smallest target values ($0 < k < 1$) from D and thus in TLAD $\hat{y} = median_{y_i \in \tilde{D}}(\{y_i\})$.

Then, the robust regression tree in this paper is defined as:

Definition 1 (Robust Regression Tree). *In a robust regression tree, an internal tree node is split so as to minimize the weighted robust loss function* $\frac{|D_L|}{|D^n|} L_l(D_L) + \frac{|D_R|}{|D^n|} L_l(D_R)$*, where* D_L *and* D_R *are two (trimmed) data subsets corresponding to the children nodes. The leaf nodes take the median of target values in the leaf node as the prediction value.*

3.2 Robust Regression Tree in the Distributed Environment

In contemporary distributed computation systems [14, 22], one node of the cluster is designated as the master processor and the others are the workers. Denote the number of workers by P. The training instance set is instance-divided into P disjoint subsets stored in different workers and each worker can only access its local data subset. Let D_p be the set of data instances stored at worker p, such that $\cup_{p=1}^{P} D_p = \mathcal{D}$. For $p, q \in \{1, \dots, P\}$, $D_p \cap D_q = \emptyset$ and $|D_p| \approxeq |\mathcal{D}|/P$. Denote the data instances in D_p belonging to a tree node n by D_p^n. A straightforward way to grow the robust regression tree layer by layer on the master is inefficient [2, 17, 22], because splitting an internal tree node requests to repeatedly access distributed data and calculate LAD (or TLAD) via expensive distributed sorting [2,22], for each trial split predicate per feature. Such a solution incurs dramatic communication and computation overheads , thereby degrading the training efficiency and scalability [2, 25].

To this end, our following proposed distributed robust regression tree will exploit data summarization [2, 17, 26], which is able to provide compact representations of the distributed data, to enhance the training efficiency.

3.3 Problem Statement

As is presented above, it is non-trivial to design an efficient training approach for distributed robust regression tree. Therefore, the problem this paper aims to solve is defined as:

Definition 2 (Training a Distributed Robust Regression Tree). *Given robust lost functions (LAD or TLAD) and training instance partitions* $D_1, \dots, D_p$ *of a data set* $\mathcal{D}$ *distributed across the workers* $1, \dots, p$ *of a cluster, training a robust regression tree in such a distributed setting involves two sub-problems: (1) to design an efficient data summarization method for the workers to extract sufficient information from local data and to transmit only such data summarization to the master with bounded communication cost. (2) to grow a robust regression tree on the master by estimating the robust loss function based on the data summarization.*

To keep things simple, we assume that all the features are discrete or categorical. However, all the discussion below can be easily generalized to continuous features [7], which is discussed in Section 6. Therefore, a split predicate on a categorical feature is a value subset. Let $\mathcal{V}_k$ represents the value set of feature k and $k \in \{1, \ldots, d\}$. For instance, given the set of data instances D^n on a tree node n and a value subset on feature k, $\mathcal{V}_k^- \subset \mathcal{V}_k$, two data subsets partitioned by $\mathcal{V}_k^-$ are $D_L = \{(\boldsymbol{x}_i, y_i)|(\boldsymbol{x}_i, y_i) \in D^n, x_{i,k} \in \mathcal{V}_k^-\}$ and $D_R = D^n - D_L$.

Often, regression tree algorithms also include a pruning phase to alleviate the problem of overfitting the training data. For the sake of simplicity, we limit our discussion to regression tree construction without pruning. However, it is relatively straightforward to modify the proposed algorithms to incorporate a variety of pruning methods [2, 7].

4 Distributed Robust Regression Tree

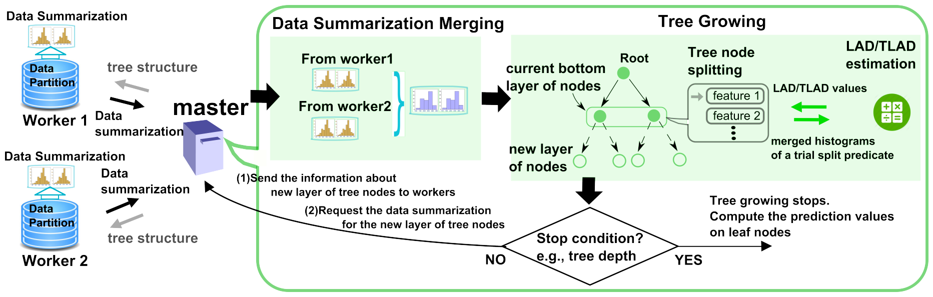
Data Summarization
Data Partition
Worker 1
tree structure
Data summarization
master
Data Summarization
Data Partition
Worker 2
Data summarization
tree structure
Data Summarization Merging
From worker1
From worker2
Tree Growing
current bottom layer of nodes
Root
Tree node splitting
feature 1
feature 2
LAD/TLAD estimation
LAD/TLAD values
merged histograms of a trial split predicate
new layer of nodes
(1)Send the information about new layer of tree nodes to workers
(2)Request the data summarization for the new layer of tree nodes
Stop condition? e.g., tree depth
NO
YES
Tree growing stops. Compute the prediction values on leaf nodes

Fig. 1. Framework of the distributed robust regression tree (best viewed in colour).

In this part, we introduce the key contribution, the distributed robust regression tree, referred to as DR2-Tree.

Overview: As is shown in Figure 1, in DR2-Tree the master grows the regression tree layer by layer in the top-down manner. Each worker retains the split predicates of the so-far trained tree nodes for data summarization. An efficient dynamic-histogram based data summarization approach is designed for workers to communicate with the master (refer to Section 4.1). Then, by using such approximate descriptions of data, the master is able to efficiently evaluate robust loss functions for determining the best split of each internal tree node, thereby circumventing expensive distributed sorting for deriving LAD/TLAD (refer to Section 4.2). Finally, the master sends the new layer of tree nodes to each worker for the next round of node splitting.

4.1 Data Summarization on Workers

Our data summarization technique adopts the dynamic histogram, a concise and effective data structure supporting mergable operations in the distributed setting [2, 26]. The one-pass nature of our proposed data summarization algorithm enables it to be adaptable to the distributed streaming learning [2] as well. Moreover, we will derive efficient robust loss function estimation algorithm based on such data summarization in the next subsection.

Algorithm 1 Data summarization on a worker

```
Input: data partition in this worker, e.g., $D_p$
Output: histogram sets $\{\mathcal{H}_p^n\}$ describing the target value distribution in internal tree node $n$
    # Bins in each histogram are maintained according to the order of bin boundaries.
    # $T_{v_r}^{n_i}$: a priority queue recording the distances between neighbouring bins.
 1: for each data sample $(\boldsymbol{x}_i, y_i)$ in $D_p$ do
 2:   search the tree built so far to locate the leaf node, e.g. $n_i$, to which sample $(\boldsymbol{x}_i, y_i)$ belongs.
 3:   for each feature value $x_{i,k}$ of $\boldsymbol{x}_i$ do
 4:     search for the bin $b_{incl}$ such that $y_i \in [b_{incl}.l, b_{incl}.r]$ by the binary search over bins of
        $H_{k,x_{i,k}}^{n_i}$
 5:     if there exits such a bin $b_{incl}$ for $y_i$ then
 6:       only update the bin $b_{incl}$ by $b_{incl}.c = b_{incl}.c + 1$, $b_{incl}.s = b_{incl}.s + y_i$
 7:     else
 8:       # $b_{lower}$ and $b_{upper}$ are obtained during the above search process for $b_{incl}$.
 9:       $b_{lower} = \underset{b_j \in \{b_k | b_k.r \leq y_i\}}{\text{argmax}} \; b_j.r$
10:       $b_{upper} = \underset{b_j \in \{b_k | b_k.l \geq y_i\}}{\text{argmin}} \; b_j.l$
11:       insert a new bin $(y_i, y_i, 1, y_i)$ into $H_{k,x_{i,k}}^{n_i}$ between bin $b_{lower}$ and $b_{upper}$
12:       insert two new neighbour-bin distances $|b_{lower}.r - y_i|$ and $|b_{upper}.l - y_i|$ to the $T_{v_r}^{n_i}$
13:       if current $|H_{k,x_{i,k}}^{n_i}| >$ histogram space bound $\beta$ then
14:         for the pair of bins $b_u$ and $b_v$ with the minimum distance in $T_{v_r}^{n_i}$, replace the bins
            $b_u$ and $b_v$ in $H_{k,x_{i,k}}^{n_i}$ by the merged bin as:
            $(\min(b_u.l, b_v.l), \max(b_u.r, b_v.r), b_u.c + b_v.c, b_u.s + b_v.s)$
15:       end if
16:     end if
17:   end for
18: end for
```

During the data summarization process, worker p builds a *histogram set* denoted by $\mathcal{H}_p^n = \{H_{r,v_r}^n\}$, for each tree node on the bottom layer, e.g., node n. It summarizes the target value distributions of D_p^n, the data instances belonging to tree node n in data partition D_p. H_{r,v_r}^n is a histogram describing the target value distribution of data instances having value v_r on feature r in D_p^n. H_{r,v_r}^n is a space bounded histogram of maximum β bins ($|H_{r,v_r}^n| \leq \beta$), e.g. $H_{r,v_r}^n = \{b_1, \ldots, b_\beta\}$. Let $count(H)$ (or $count(\mathcal{H})$) be the number of data instances summarized by a histogram H (or a histogram set $\mathcal{H}$). Each bin of a histogram is represented by a quad, e.g. $b_i = (l, r, c, s)$, where l and r are the minimum and maximum target values in this bin, c is the number of target values falling under this bin and s is the sum of the target values. We will see how such quad elements are used in growing the tree in the next subsection. The number of bins β in the histograms is specified through a trade-off between accuracy and computational and communication costs: a large number of bins gives a more accurate data summarization, whereas small histograms are beneficial for avoiding time, memory, and communications overloads.

Algorithm 1 presents the data summarization procedure on each worker, which updates the local data instances one by one to the corresponding histogram set. First, the tree node n_i in the bottom layer of the tree for a data instance $(\boldsymbol{x}_i, y_i) \in D_p$ is found (line $1-2$) and its associated $\mathcal{H}_p^{n_i}$ will be updated. For each feature value of $(\boldsymbol{x}_i, y_i)$, y_i is inserted to the corresponding histogram in $\mathcal{H}_p^{n_i}$ by either updating an existing bin having the value range covering y_i (line $3-6$) or inserting a new bin $(y_i, y_i, 1, y_i)$ to the histogram (line $7-12$). Second, if the size of the histogram exceeds the predefined maximum value β then the nearest bins are continuously merged until addressing the limit β (line $13-16$). A temporary priority structure (e.g., $T_{v_r}^{n_i}$) is maintained for efficiently finding closest bins to merge (line $13-16$). Finally, workers only send such data summarization to the master.

Complexity Analysis: In line $2-6$, the binary search over bins of a histogram takes $\log \beta$ time. Then the priority structure can support in finding the nearest bins and updating bin distances in $\log \beta$ time (line $13-16$). Overall, the time complexity of Algorithm 1 is $\mathcal{O}(|D_p| d \log \beta)$. Compared with the histogram building approach in [2, 26], our method circumvents the sorting operation for updating individual data instances and improves the efficiency, as is demonstrated in Section 5. The communication complexity for transmitting data summarization of the bottom layer of nodes between the worker and master is bounded by $\mathcal{O}(\max_r(|\mathcal{V}_r|) d\beta)$ independent of the size of the data partitions. For the features with high cardinality, our data summarization can incorporate extra histograms over feature values to decorrelate the communication cost and the feature cardinality [2, 26].

4.2 Tree Growing on the Master

In this part, we will first outline the tree node splitting process using the data summarization in growing the tree. Then, we present the involved two fundamental operations in detail, namely the histogram merging and LAD/TLAD estimation.

Tree Node Splitting: In order to find the best split of a tree node, we need a histogram set summarizing all the data instances falling under this node. Therefore, as is presented

```
Algorithm 2 Tree node splitting
Input: histogram sets of tree node $n$ from all data partitions, $\mathcal{H}_1^n, \dots, \mathcal{H}_P^n$.
Output: the split feature and associated value set for tree node $n$.
1: build a unified histogram set summarizing the overall target value distribution for this tree
   node $\mathcal{H}^n = merge(\mathcal{H}_1^n, \dots, \mathcal{H}_P^n)$ by using the histogram merging operation presented in
   Algorithm 3
2: for each feature $k \in \{1, \dots, d\}$ do
3:     Sort the feature values in $\mathcal{V}_k$ according to the median estimations of data in the corre-
       sponding histograms [25].
4:     $\tilde{\mathcal{V}}_k$: the sorted feature values in $\mathcal{V}_k$.
5:     iterate over $\tilde{\mathcal{V}}_k$ to find a $v_j$ and the associated feature value subsets $\mathcal{V}^- = \{v_j | j \leq i\}$ and
       $\mathcal{V}^+ = \mathcal{V}_k - \mathcal{V}^-$, so as to minimize the weighted robust loss function.
6: end for
7: return the feature and value subsets, which achieve the minimum robust loss.
```

in Algorithm 2, a unified histogram set is built by using the histogram merging operation, which will be described in Algorithm 3. Then, it iterates over each feature to find a split predicate (Line 4-6), i.e., a feature value subset, so as to minimize the weighted loss as:

$$\{v*, \mathcal{V}^{+*}, \mathcal{V}^{-*}\} = \underset{v_i, \mathcal{V}^+, \mathcal{V}^-}{argmin} \hat{L}_l(H^-) \frac{count(H^-)}{count(\mathcal{H}^n)} + \hat{L}_l(H^+) \frac{count(H^+)}{count(\mathcal{H}^n)} \quad (1)$$

where $\hat{L}_l(\cdot)$ is the histogram based estimation of robust loss functions (LAD/TLAD), which is presented in Algorithm 4. For a trial feature value subset, e.g. $\mathcal{V}^- = \{v_j | j \leq i\}$ and $\mathcal{V}^+$, we need to estimate the LAD/TLAD over the data subsets defined by $\mathcal{V}^-$ and $\mathcal{V}^+$. Therefore, two temporary histograms, e.g., H^- and H^+ are built by merging the histograms in $\mathcal{H}^n$ corresponding to the feature values present in $\mathcal{V}^-$ and $\mathcal{V}^+$, i.e., $H^- = merge(\{H_{v_j}^n | j \leq i\})$ and $H^+ = merge(\{H_{v_j}^n | j > i\})$ approximating the distributions of two data subsets defined by $\mathcal{V}^-$ and $\mathcal{V}^+$.

Finally, when the tree reaches the stopping depth, the predictions on the leaf nodes can be exactly derived by accessing the distributed dataset. This step is only performed when the tree growing phase is finished.

Histogram Merging: Our proposed histogram merging operation is a one-pass method over the bins of histograms and creates a histogram summarizing the union of data distribution of the two histograms. As is presented in Algorithm 2, it is mainly used in two cases: (1) build a unified histogram set for each tree node on the bottom layer; (2) build temporary histograms to approximate the target value distributions of two data subsets defined by a trial feature value subset. Algorithm 3 presents the histogram merging algorithm. Two histograms H_1 and H_2 are first combined in the merge-sort way. During this process, a heap is maintained to record the neighbour-bin distances. Then, bins which are closest are merged together to form a single bin. The process repeats until the histogram has β bins.

Algorithm 3 Histogram merging

```
Input: Two histograms, e.g., $H_1$ and $H_2$.
Output: A histogram $H$ summarizing the union of data distribution in $H_1$ and $H_2$.
    # $E$ is a priority queue recording the distances between neighbouring bins.
 1: $H$: merged histogram.
 2: while $H_1$ and $H_2$ have bins do
 3:     $b_i$ and $b_j$: current popped bins from $H_1$ and $H_2$
 4:     if $b_i.l < b_j.l$ then
 5:         insert $b_i$ in $H$
 6:     else
 7:         insert $b_j$ in $H$
 8:     end if
 9:     insert the new neighbour bin distance in $E$.
10: end while
11: insert the remaining bins in $H_1$ or $H_2$ to $H$.
12: while $|H| >$ histogram space bound $\beta$ do
13:     pop from $E$ the pair of bins $b_u$ and $b_v$ with the minimum bin distance
14:     replace the bins $b_u$ and $b_v$ in $H$ by the merged bin
        $(\min(b_u.l, b_v.l), \max(b_u.r, b_v.r), b_u.c + b_v.c, b_u.s + b_v.s)$
15: end while
```

Complexity Analysis: In Algorithm 2 Line 2-11 scans the bins in histograms H_1 and H_2 once and thus takes $\mathcal{O}(\beta)$. Line 12-14 combines the redundant bins by using the heap, which takes $\mathcal{O}(\beta log(\beta))$.

LAD/TLAD Estimation: A straightforward method to estimate LAD (or TLAD) based on a histogram is to first make a median estimate and then to sample data in each bin of the histogram to approximate individual absolute deviations [2, 26]. Both the median estimation and data sampling process introduce errors into the LAD (or TLAD) estimation [25].

To this end, we propose a more efficient and precise algorithm to approximate LAD and TLAD in one-pass way. Before giving the details, we first define some notations.

Definition 3 (Quantile Bin of a Histogram). *Given a histogram $H = \{b_1, \dots, b_\beta\}$, $count(H)$ the number of values this histogram summarizes and a quantile q over the summarized values, the quantile bin b_q addresses $\sum_{b_i<b_q} b_i.c < count(H) \cdot q$ and $\sum_{b_i \le b_q} b_i.c \ge count(H) \cdot q$*

Definition 4 (R-Partial-Sum of a Bin). *Given a bin $b = (l, r, c, s)$ of a histogram, R-Partial-Sum of bin b, $S_p(b, R)$ is defined as the sum of the R smallest values summarized in this bin.*

Recall that in the data summarization in Algorithm 1, the histogram updating process unites neighbouring bins according to the distance of bin boundaries. This allows the bins to adapt to the data distribution. Regarding the values summarized by a bin in a histogram (e.g., b), we can safely assume that they are uniformly distributed in range $[b.l, b.r]$ [2]. Therefore, we provide the lemma below, which will be used for LAD estimation, to approximate R-Partial-Sum:

Algorithm 4 LAD / TLAD Estimation

```
Input: A histogram $H$, trim ratio $\tau$
Output: LAD or TLAD estimation.
1: $s$: variable to record $\sum_{b_i > b_m} b_i.s - \sum_{b_i < b_m} b_i.s$ , $s_t$: variable to record $\sum_{b_i > b_{\overline{q}}} b_i.s - \sum_{b_q < b_i < b_m} b_i.s$.
   # $m$ is the index of $1/2$-quantile bin; $q$ and $\overline{q}$ are the indices of $\tau$-quantile and (1-$\tau$)-quantile bins.
2: $c$: variable to record the current count of data instances.
3: for each bin $b_i$ in $H$ do
4:    $c = c + b_i.c$
5:    record quantile bin $m$, $q$ and $\overline{q}$ when $c$ address the corresponding conditions.
6:    if 0.5-quantile bin $m$ is found then
7:       $s = s + b_i.s$
8:    else
9:       $s = s - b_i.s$
10:   end if
11:   if $\tau$-quantile bin $b_q$ is found and $1/2$-quantile bin $b_m$ is not found then
12:      $s_t = s_t - b_i.s$
13:   else if (1-$\tau$)-quantile bin $b_{\overline{q}}$ is not found and $1/2$-quantile bin $q$ is found then
14:      $s_t = s_t + b_i.s$
15:   end if
16: end for
17: LAD: $\hat{L}_l(H) = s + b_m.s - 2\hat{S}_p(b_m, R)$,
18: TLAD: $\hat{L}_l(H, \tau) = s_t + \hat{S}_p(b_q, R_1) - b_q.s + \hat{S}_p(b_{\overline{q}}, R_2) + b_m.s - 2\hat{S}_p(b_m, R)$
```

Lemma 1. *For a bin $b = (l, r, c, s)$ of a histogram and an integer R ($R \leq b.c$), under the assumption of the uniform distribution of values in the bin, R-Partial-Sum of bin b can be approximated by* $S_p(b, R) \approxeq \hat{S}_p(b, R) = \begin{cases} b.s & : R = b.c \\ R \cdot b.l + R(R-1)\delta & : otherwise \end{cases}$, *where* $\delta = \frac{(b.s - b.r - b.c \cdot b.l + b.l)}{(b.c-2)(b.c-1)}$.

Proof.

Now we provide the following lemma for estimating the LAD/TLAD based on a histogram as:

Lemma 2. *Given a histogram $H = \{b_1, \ldots, b_\beta\}$, the LAD/TLAD over the data summarized by histogram H can be exactly computed by:*

(1) $L_l(H) = \sum_{b_i > b_m} b_i.s - \sum_{b_i < b_m} b_i.s + b_m.s - 2S_p(b_m, R)$, *where* $R = \lceil \frac{C}{2} \rceil - \sum_{b_i < b_m} b_i.c$, $C = count(H)$ *is the total number of data instances covered in histogram* H, *and an* b_m *is the* $\frac{1}{2}$*-quantile bin.*

(2) $L_l(H, \tau) = \sum_{b_m < b_i < b_{\overline{q}}} b_i.s - \sum_{b_q < b_i < b_m} b_i.s + S_p(b_q, R_1) - b_q.s + S_p(b_{\overline{q}}, R_2) + b_m.s - 2S_p(b_m, R)$, *where* b_m, b_q *and* $b_{\overline{q}}$ *are respectively the* $\frac{1}{2}$, τ *and* $(1-\tau)$*-quantile bins,* $R = \lceil \frac{C}{2} \rceil - \sum_{b_i < b_m} b_i.c$, $R_1 = C \cdot \tau - \sum_{b_i < b_q} b_i.c$, $R_2 = C \cdot (1-\tau) - \sum_{b_i < \overline{q}} b_i.c$.

Proof. Limited by the space, refer to [30] for the proof details.

Lemma 2 suggests that in estimating LAD/TLAD based on a histogram, the median estimation step is circumvented. Meanwhile, given the histograms LAD/TLAD can be estimated through replacing $S_p(\cdot)$ in Lemma 2 by $\hat{S}_p(\cdot)$ defined in Lemma 1 and exactly computing the remaining terms. In summary, the histogram based estimation of robust loss functions can be expressed as:

$$\hat{L}_l(H) = \sum_{b_i > b_m} b_i.s - \sum_{b_i < b_m} b_i.s + b_m.s - 2\hat{S}_p(b_m, R) \tag{2}$$

and

$$\begin{aligned}\hat{L}_l(H,\tau) = \sum_{b_m < b_i < b_{\overline{q}}} b_i.s - \sum_{b_q < b_i < b_m} b_i.s + \hat{S}_p(b_q, R_1) - b_q.s + \\ \hat{S}_p(b_{\overline{q}}, R_2) + b_m.s - 2\hat{S}_p(b_m, R)\end{aligned} \tag{3}$$

where R, R_1 and R_2 are defined as Lemma 2.

On the basis of Lemma 2, we put forward the LAD/TLAD estimation algorithm, as is shown in Algorithm 3. It is able to estimate LAD or TLAD in one-pass over the bins of the given histogram. In our LAD/TLAD estimation algorithm, the only approximate part is $\hat{S}_p(b, R)$. Now we provide the theoretical error bound on it:

Theorem 1. *Given a bin $b = (l, r, c, s)$ of a histogram and R ($R \leq b.c$), if $R = 1$ or $R = b.c$, $\hat{S}_p(b, R)$ provided in Lemma 1 is the exact R-Partial-Sum. Otherwise, the approximation error of R-Partial-Sum of bin b, $S_p(b, R) - \hat{S}_p(b, R)$ is bounded within $[(R - b.c) \cdot (b.r - b.l), b.s - b.c \cdot b.l]$.*

Proof.

Therefore, the LAD/TLAD estimation has bounded errors as well.

5 Experimental Evaluations

In this section, we perform extensive experiments to demonstrate the efficiency and effectiveness of DR2-Tree. we first present the setup of the experiments including datasets, baselines and the implementation environment. Then, we report the results focusing on three aspects: the efficiency in terms of training time and speedup, the effectiveness in terms of prediction accuracy and the data summarization performance in DR2-Tree.

5.1 Setup

Dataset: In the experiments, we use one synthetic and two real datasets.

Synthetic Data: Our synthetic data generator[4] produces data instances with specified number of features. For each distinct feature value combination e.g., $(v^1, \ldots, v^d)$, where v^1 is the value of the first feature and d is the number of features, it generates several data instances having such feature values and the target values sampled from a Gaussian distribution. Such Gaussian distributions are specific *w.r.t.* feature-value combinations. Meanwhile, data instances with outliers on the target variable are injected based on a Bernoulli distribution. The probability of the Bernoulli distribution is specified through the *percentage of outliers* in the produced dataset and it is set as 0.05 initially, i.e., 5% of data instances have outlier target values. *The magnitude of outlier target values* is defined as the times of the Gaussian distribution mean. By default, the magnitude is 3, which means that the target value of an outlier data instance is sampled from a Gaussian distribution with 3 times larger mean than the mean of the corresponding feature value combination's distribution. The percentage and magnitude of outliers will be tuned later in Subsection 5.3.

Flight Dataset: It contains the scheduled and actual departure and arrival times of flights reported by certified U.S. air carriers from 1987-2008 [5]. It contains data instances with abnormal values on the "arrival delay" and "departure delay" attributes, due to abnormal events, e.g., weather, security, etc. In our experiments, we use the attribute "ArrDelay" as the target variable and the categorical features as the independent variables. The cardinalities of these categorical features vary from 10 to 1032.

Network Dataset: It is a dataset provided by a major European telecommunication service provider consisting of active measurements from probes within a residential ISP network. The probes measure various performance fields such as the throughput, jitter and delay between their location and chosen end-points. Furthermore, each probe and end-point are associated with various categorical and continuous features, such as the time of the measurement, the location of the endpoints and the configuration of the lines. Finally the tests cover a period of 2 days and involve 124 probes and 1314 targets. This dataset is noisy in the sense that due to network anomalies and events, the measurements could have huge outlier values.

Baselines: ER2T is a distributed robust regression tree. SRT and DHRT are two representative distributed regression trees in the literature [2, 17, 26]. ER2T: It refers to the **e**xact distributed **r**obust **r**egression **t**ree. It builds the robust regression tree on the master by exactly calculating the robust loss functions in a distributed way. SRT: It refers to the distributed regression tree based on square error criteria [17] in Apache Spark machine learning tool set [6]. Prior to the tree induction, a pre-processing step is performed to obtain static and equidepth histograms for each feature and the split points are constantly selected from the bins of such histograms in the training phase. DHRT: It implements a single distributed regression tree based on [26], which employs **d**ynamic **h**istograms [2] to summarize statistics in distributed data for evaluating the square error split criterion on the master. In building histograms, it requires to sort the bins each time a data instance is added to the set already represented by the histogram [2].

[4] `https://github.com/weilai0980/DRSquare_tree/tree/master/dr2tree_src`

[5] `http://stat-computing.org/dataexpo/2009/the-data.html`

[6] `http://spark.apache.org/docs/latest/mllib-decision-tree.html`

In Subsection 5.3, we will also use random forests (RF) and gradient boosted regression trees (GBT) in the distributed machine learning library Spark MLlib to compare with our robust regression tree in terms of accuracy.

Implementation: Our proposed DR2-Tree and baselines are all implemented on Apache Spark, a popular distributed data processing engine. The engine is deployed on a cluster of 23 servers, each with 16 cores (2.8GHz) and 32G of RAM.

5.2 Efficiency

In this group of experiments, we evaluate the efficiency of growing regression trees under different conditions. The training time is measured as the total time for growing a tree from the root node until the specified depth. To mitigate the effects of varying cluster conditions, all the results have been averaged over multiple runs.

We consider four parameters to tune in this set of experiments, namely the tree depth, training data size, maximum number of bins in data summarization and the number of workers. They have direct effect on the training time [2, 17, 22, 26]. The experiments are performed by varying one parameter while keeping the others as default values. By default, the maximum number of bins is set as 500, the number of workers is 5, the depth is 6 and the size of the training dataset is 10 million initially.

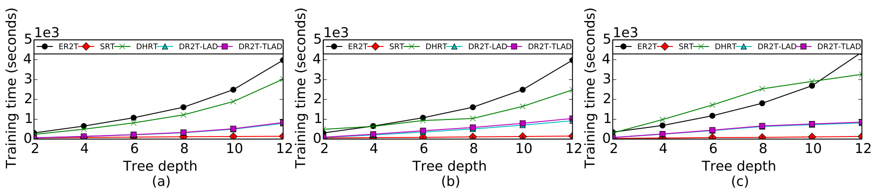

Fig. 2. Training time *w.r.t.* the depth of the tree. (a) synthetic dataset (b) flight dataset (c) network dataset. (best viewed in colour)

Depth: Figure 2 presents the training time as a function of the training depth of the regression tree. DR2-Tree outperforms ER2T and DHRT by $3\times$ and $2\times$ faster in average. In ER2T the training time consistently takes the longest, as it computes the expensive exact median and LAD (TLAD) in the distributed setting. SRT takes 0.5 times less time than DR2-Tree. This is because SRT constantly summarizes the data using fixed bins and thus takes less time to extract statistics in bins from distributed data during the training phase. But square-error based SRT and DHRT are less robust to noisy data than DR2-Tree, which will be shown in the next subsection.

Data size: In Figure 3, we present the training time as a function of the size of the training dataset, i.e., the number of data instances. Due to the one-pass nature of data summarization and tree growing processes, the training time of DR2-Tree increases linearly, highlighting the scalability. DHRT has a quickly increasing training time in part due to the quadratic computation in updating histograms. DR2-Tree takes $3\times$ and $2\times$ less training time in average than ER2T and DHRT.

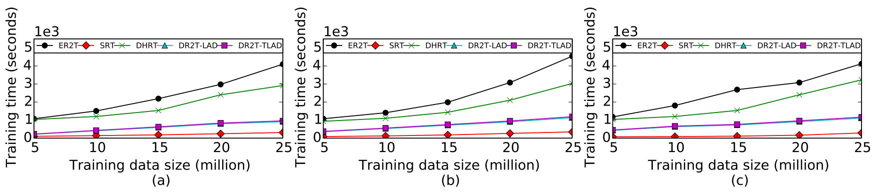

Fig. 3. Training time *w.r.t.* the size of training dataset. (a) synthetic dataset (b) flight dataset (c) network dataset. (best viewed in colour)

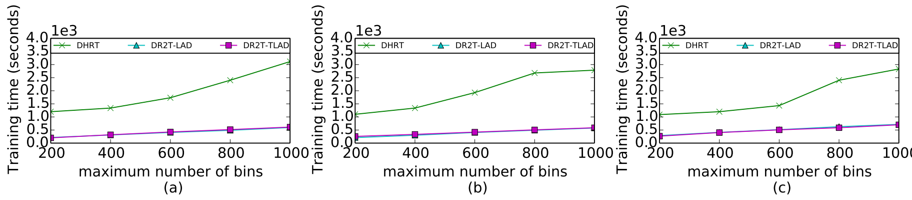

Fig. 4. Training time *w.r.t.* the maximum number of bins in data summarization. (a) synthetic dataset (b) flight dataset (c) network dataset. (best viewed in colour)

Maximum number of bins: In Figure 4, we investigate the effect of the maximum number of bins in data summarization on the training time. ER2T employs no data summarization. In SRT, bins are built according to the cardinality of features in data. Therefore, varying the number of bins has no effect on ER2T and SRT and in Figure 4 only the results of DR2-Tree and DHRT are reported. The number of bins affects both the efficiency of data summarization on workers and tree growing on the master, and thus in general the training time is positively correlated with the maximum number of bins. At the highest level of bin numbers, the training time of DR2-Tree is average 4 times less than DHRT.

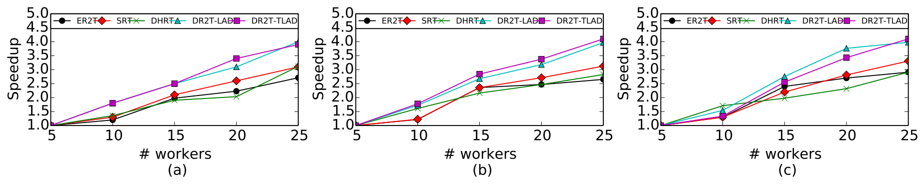

Fig. 5. Training time *w.r.t.* the number of workers. (a) synthetic dataset (b) flight dataset (c) network dataset. (best viewed in colour)

Number of workers: In Figure 5, we proceed to investigate the speedup for different numbers of workers. For large datasets, the communication between workers and the master is negligible relative to the gain in the data summarization building phase. Therefore, increasing the number of workers is beneficial for speeding up the training process [2]. DR2-Tree presents $2\times$ higher speedup than ER2T at the highest level of number of workers.

5.3 Effectiveness

In this part, we evaluate the prediction accuracy of DR2-Tree and baselines under different dataset properties and regression tree set-ups. Specifically, we aim to study the effect of the maximum number of bins, the outlier percentage and magnitude in the training dataset. The prediction accuracy is measured by normalized root mean square error (NRMSE), so as to facilitate the comparison between datasets. Lower values of NRMSE are considered better. The trim ratio in DR2-Tree-TLAD is chosen within the range of $[0.05, 0.15]$ by cross-validation in this group of experiments. The un-tuned parameters in each group of experiments are set as the default values in Section 5.2.

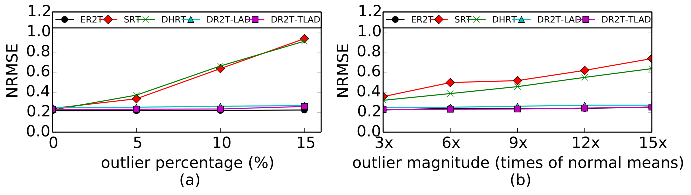

Fig. 6. Prediction accuracy *w.r.t.* the (a) outlier percentage and (b) magnitude in the training dataset (best viewed in colour).

Outlier properties in the dataset: In this group of experiments, we investigate the effect of the noise level of the training dataset, namely the outlier percentage and magnitude, on the prediction accuracy. Since we can only manipulate the noise level of the synthetic dataset, only the results on the synthetic dataset are reported in Figure 6.

In Figure 6(a), we increase the percentage of data instances with outlier target values while keeping the magnitude of the outlier values as $3\times$ of the target value mean. It is observed that initially when the training dataset has no outliers, the accuracies of all approaches are highly close. As the percentage of outliers increases, the accuracy difference between the square error and robust error criterion based approaches becomes significant.

In Figure 6(b), we study the effect of the outlier magnitude on the accuracy. In this group of experiments, 5% data instances have outlier target values. When the magnitude of outliers increases, ER2T and DR2-Tree demonstrate stable accuracy and have 2 times less errors than SRT and DHRT at the highest level of the outlier magnitude. Compared with Figure 6(a), we also observe that square error based approaches are more sensitive to the outlier percentage than to the outlier magnitude in the dataset.

Maximum number of bins:

Figure 7 displays the prediction accuracy for different number of bins in the data summarization. As is presented in Section 4, the number of bins affects the precision

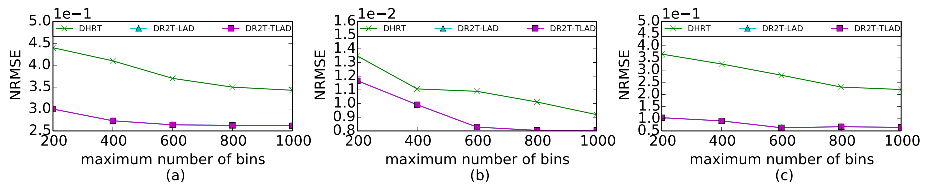

Fig. 7. Prediction accuracy *w.r.t.* the maximum number of bins in data summarization. (a) synthetic dataset (b) flight dataset (c) network dataset. (best viewed in colour)

of error criterion estimation in DR2-Tree. Meanwhile, it should avoid setting the number of bins too large, otherwise the training efficiency would degrade, as is shown in Figure 4. Since only DR2-Tree and DHRT have tunable dynamic histograms, only the results of them are shown.

For the noisy synthetic and network datasets, as the number of bins increases, the master in DR2-Tree can obtain more precise data summarization based LAD/TLAD estimation and thus yields decreasing prediction errors. DR2-Tree outperforms DHRT by around 3 times. For the flight data, they present comparable accuracies.

Table 1. Overall accuracy comparison (NRMSE).

datasets	ER2T	SRT	DHRT	DR2-Tree-LAD	DR2-Tree-TLAD	RF	GBT
Synthetic data	0.225	0.481	0.493	0.224	0.219	0.481	0.476
Flight data	0.00882	0.00874	0.00908	0.00889	0.00890	0.00836	0.00835
Network data	0.061	0.148	0.153	0.0629	0.0581	0.145	0.181

Overall accuracy comparison: In this part, we run all the approaches by cross-validation to achieve the best prediction accuracies for the three datasets and report the results in Table 1. The synthetic dataset is set to have 5% outlier target values with magnitude 3. It shows that robust error criterion based approaches have around 50% less error than square error based approaches, i.e., SRT, DHRT, RF and GBT. For not so noisy data, i.e. the flight data, two types of approaches have very comparable accuracy. Such results also demonstrate the wide applicability of our DR2-Tree.

5.4 Data Summarization Performance in DR2-Tree

Communication cost of data summarization: Table 2 shows the average communication cost of data summarization for a tree node between the worker and master. It is measured by the amount of basic data units (e.g., integer, floats, etc.). We study the communication cost variation as tree grows, since DR2-Tree employs dynamic histograms, which are rebuilt for the new-grown bottom layer of tree nodes.

From Table 2, we observe that the communication cost increases first, then keeps relatively constant or slightly decreases, as the tree grows. This is because initially the data instances are assigned to small number of tree nodes and each tree node has data summarization with histograms of full bins to summarize the data. As the tree grows and has more nodes, the amount of data instances in each node is decreasing and the data summarization requires less bins in histograms and therefore the communication cost keeps relatively stable or decreases. The communication cost is different across datasets. This is because the number of features and the size of feature value set are different for the three datasets.

Table 2. Data summarization communication cost as the tree grows.

Tree depth	2	4	6	8	10
Synthetic data	12500	25000	25000	18359	17871
Flight data	560000	1120000	1120000	1225000	1050000
Network data	200000	400000	437500	412500	411500

Accuracy of data summarization: In this part, we proceed to further investigate the accuracy of the data summarization based LAD/TLAD estimation in DR2-Tree. In order to understand the accuracy under different data distributions [2], we generate seven synthetic sets via different kinds of probability distributions and summarize each data set by using the histogram building approach in DR2-Tree. The probability distributions and associated parameters are listed in Table 3. For each distribution, 10^5 data points are generated. Then, we compare the histogram based LAD/TLAD estimation with the exact values by using the mean absolute percentage error. It expresses the accuracy as a percentage and fits for the datasets of different value scales. We run the process of data generation, histogram building and LAD/TLAD estimation for several times to report average results in Table 3.

We observe that the error significantly decreases as the number of bins in histograms is increased. For 800 number of bins, most of the errors are below 0.1. The use of a memory-bounded data summarization in our framework naturally comes at a cost in accuracy. When the data distribution is highly skewed, practitioners can apply alternative R-partial sum approximate estimation based on the assumed distribution. This only changes the LAD/TLAD estimation component while keeping the training framework intact.

6 Discussion

Our current version of DR2-Tree focuses on the robust regression with categorical features. It can be smoothly extended to handle numeric or mixed features. For numeric features, besides the histograms built on the target values in current DR2-Tree, we can integrate additional histograms on the domains of numeric features [2, 17, 26] to form two-dimensional histogram based data summarization, such that these histograms respectively provide split candidates and error criterion estimation. Meanwhile, our data summarization has potential to support alternative robust error criterion such as Huber loss in the future. Lastly, note that our robust regression tree learner also supports binary (0-1) classification tasks by modeling them as instances of logistic regression.

Table 3. LAD/TLAD estimation error *w.r.t.* the maximum number of bins in histograms (two numbers in each cell respectively correspond to the mean absolute percentage errors of LAD and TLAD.)

#Bins / Distribution	200	400	600	800
Uniform ($[0, 100]$)	(0.0000410, 0.347)	(0.0000411, 0.216)	(0.0000393, 0.122)	(0.0000392, 0.029)
Normal ($\mu = 0, \sigma = 1$)	(0.571, 0.844)	(0.450, 0.690)	(0.173, 0.403)	(0.0712,0.213)
Exponential($\mu = 0.5$)	(0.243, 0.0753)	(0.221, 0.0776)	(0.205,0.0761)	(0.143, 0.0678)
Beta ($a = 0.5, b = 0.5$)	(0.0000316, 0.198)	(0.0000309, 0.123)	(0.0000315, 0.0721)	(0.0000313, 0.0612)
Gama ($a = 3, b = 1$)	(0.118, 0.381)	(0.0909, 0.184)	(0.0890, 0.114)	(0.0772, 0.0983)
Lognormal ($\mu = 1, \sigma = 0.5$)	(0.138, 0.201)	(0.0940, 0.135)	(0.0862, 0.101)	(0.0723, 0.0975)
Chisquare ($v = 10$)	(0.243, 0.130)	(0.196, 0.115)	(0.138, 0.102)	(0.078, 0.083)

7 Conclusion

In this paper, we propose an efficient distributed robust regression tree for handling large and noisy data. Our DR2-Tree employs a novel data summarization technique, which is able to support both distributed information extraction and robust error criterion estimation for growing the regression tree. Extensive experiments reveal that: (1) Our proposed DR2-Tree is robust to datasets with various outlier percentages and magnitudes. (2) DR2-Tree exhibits comparable accuracy as the conventional distributed regression tree for relatively clean datasets with rare outliers. (3) DR2-Tree is much more efficient than exact robust regression and the dynamic histogram based regression tree [2, 26]. Such results verifies the broad applicability of our DR2-Tree framework.

8 Acknowledgements

The authors thank Mathias Niepert for his insightful input to this work. The research leading to these results has received funding from the European Union's Horizon 2020 innovation action program under grant agreement No 653449-TYPES as well as Nano-Tera.ch through the OpenSense2 project.

References

1. Basilico J D, Munson M A, Kolda T G, et al. COMET: A recipe for learning and using large ensembles on massive data. In: IEEE ICDM, 2011: 41-50.
2. Ben-Haim Y, Tom-Tov E. A streaming parallel decision tree algorithm. In: JMLR, 2010, 11: 849-872.
3. Chm-les X L C, CA O, Yan R J. Decision tree with better ranking. In: AAAI 2003.
4. Darlington J, Guo Y, Sutiwaraphun J, et al. Parallel induction algorithms for data mining. In: Advances in Intelligent Data Analysis Reasoning about Data. Springer Berlin Heidelberg, 1997: 437-445.
5. Esposito F, Malerba D, Semeraro G, et al. A comparative analysis of methods for pruning decision trees. In: IEEE TPAMI, 1997. 19(5): p. 476-491.

6. Freitas A A, Lavington S H. Mining very large databases with parallel processing. In: Springer Science Business Media, 1998.
7. Hastie T, Tibshirani R, Friedman J, et al. The elements of statistical learning: data mining, inference and prediction. The Mathematical Intelligencer, 2005, 27(2): 83-85.
8. Janikow C Z. Fuzzy decision trees: issues and methods. In: IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, 1998, 28(1): 1-14.
9. Jin R, Agrawal G. Communication and Memory Efficient Parallel Decision Tree Construction. In: SIAM SDM, 2003.
10. John G H. Robust Decision Trees: Removing Outliers from Databases. In: KDD, 1995: 174-179.
11. Joshi M V, Karypis G, Kumar V. ScalParC: A new scalable and efficient parallel classification algorithm for mining large datasets. In: the 12th International Parallel Processing Symposium, 1998:573579.
12. Katz G, Shabtai A, Rokach L, et al. ConfDTree: Improving decision trees using confidence intervals. In: IEEE ICDM, 2012: 339-348.
13. Koufakou A, Georgiopoulos M. A fast outlier detection strategy for distributed high-dimensional data sets with mixed attributes. In: Data Mining and Knowledge Discovery, 2010, 20(2): 259-289.
14. Lee K H, Lee Y J, Choi H, et al. Parallel data processing with MapReduce: a survey. In: ACM SIGMOD Record, 2012, 40(4): 11-20.
15. Liaw A, Wiener M. Classification and regression by randomForest. R news, 2002, 2(3): 18-22.
16. Olaru C, Wehenkel L. A complete fuzzy decision tree technique. In: Fuzzy sets and systems, 2003, 138(2): 221-254.
17. Panda B, Herbach J S, Basu S, et al. Planet: massively parallel learning of tree ensembles with mapreduce. In: VLDB Endowment, 2009, 2(2): 1426-1437.
18. Provost F, Domingos P. Well-trained PETs: Improving probability estimation trees. Technical Report CDER 00-04-IS, 2001.
19. Quinlan J R. Bagging, boosting, and C4. 5. In: AAAI/IAAI, 1996: 725-730.
20. Rodriguez J J, Kuncheva L I, Alonso C J. Rotation forest: A new classifier ensemble method. In: IEEE TPAMI, 2006, 28(10): 1619-1630.
21. Shafer J, Agrawal R, Mehta M. SPRINT: A scalable parallel classier for data mining. In: PVLDB. 1996: 544-555.
22. Shanahan J G, Dai L. Large scale distributed data science using apache spark. In: ACM SIGKDD. 2015: 2323-2324.
23. Srivastava A, Han E H, Kumar V, et al. Parallel formulations of decision-tree classification algorithms. In: Springer US, 2002.
24. Stuart C. Robust regression. Department of Mathematical Sciences, Durham University, 2011, 169.
25. Torgo L F R A. Inductive learning of tree-based regression models. Thesis 1999.
26. Tyree S, Weinberger K Q, Agrawal K, et al. Parallel boosted regression trees for web search ranking. In: ACM WWW, 2011: 387-396.
27. Wang P, Sun W, Yin D, et al. Robust tree-based causal inference for complex ad effectiveness analysis. In: ACM WSDM, 2015: 67-76.
28. Ye J, Chow J H, Chen J, et al. Stochastic gradient boosted distributed decision trees. In: ACM CIKM, 2009: 2061-2064.
29. Zadrozny B, Elkan C. Obtaining calibrated probability estimates from decision trees and naive Bayesian classifiers. In: ICML, 2001, 1: 609-616.
30. Supplementary Material. `https://infoscience.epfl.ch/record/218970`

9 Appendixes

9.1 Proof of lemmas and theorems

Lemma 1. *For a bin $b = (l, r, c, s)$ of a histogram and an integer R ($R \leq b.c$), under the assumption of the uniform distribution of values in the bin, R-Partial-Sum of bin b can be approximated by* $S_p(b, R) \approxeq \hat{S}_p(b, R) = \begin{cases} b.s & : R = b.c \\ R \cdot b.l + R(R-1)\delta : otherwise \end{cases}$, *where* $\delta = \frac{(b.s - b.r - b.c \cdot b.l + b.l)}{(b.c-2)(b.c-1)}$.

Proof. We assume that the values in a bin (e.g., b) are uniformly distributed in the value range $[b.l, b.r]$. Then, since the sum of values in the bin is given by $b.s$, we can obtain the sum of values between $b.l$ and $b.r$ is $b.s - b.r - b.l$. Meanwhile, the values in $(b.l, b.r)$ can be derived by adding incremental values over $b.l$. Therefore, the increments are considered as an arithmetic series whose elements are the products of the sequence term (e.g., $1, \ldots, b.c - 1$) and the unit increment value. The sum of such increments is $(b.s - b.r - b.c \cdot b.l + b.l)$. The unit increment value is calculated as $\frac{(b.s-b.r-b.c \cdot b.l+b.l)}{(b.c-2)(b.c-1)/2}$, where $(b.c-2)(b.c-1)/2$ is the number of unit increments derived by the sum of the arithmetic series. As a result, the R-partial sum is approximated by the sum of the sequence of the first R increments plus the base value $b.l$, i.e., $R \cdot b.l + R(R-1) \cdot \frac{(b.s-b.r-b.c \cdot b.l+b.l)}{(b.c-2)(b.c-1)}$.

Lemma 2. *Given a histogram $H = \{b_1, \ldots, b_\beta\}$, the LAD/TLAD over the data summarized by histogram H can be exactly computed by:*
(1) $L_l(H) = \sum_{b_i > b_m} b_i.s - \sum_{b_i < b_m} b_i.s + b_m.s - 2S_p(b_m, R)$, *where* $R = \lceil \frac{C}{2} \rceil - \sum_{b_i < b_m} b_i.c$, $C = count(H)$ *is the total number of data instances covered in histogram H, and an b_m is the $\frac{1}{2}$-quantile bin.*
(2) $L_l(H, \tau) = \sum_{b_m < b_i < b_{\overline{q}}} b_i.s - \sum_{b_q < b_i < b_m} b_i.s + S_p(b_q, R_1) - b_q.s + S_p(b_{\overline{q}}, R_2) + b_m.s - 2S_p(b_m, R)$, *where b_m, b_q and $b_{\overline{q}}$ are respectively the $\frac{1}{2}$, τ and $(1-\tau)$-quantile bins,* $R = \lceil \frac{C}{2} \rceil - \sum_{b_i < b_m} b_i.c$, $R_1 = C \cdot \tau - \sum_{b_i < b_q} b_i.c$, $R_2 = C \cdot (1-\tau) - \sum_{b_i < \overline{q}} b_i.c$.

Proof. Assume that the target values of data instances summarized by histogram H are given as a sorted list y_i and $C = count(H)$ is even. The median is $m = \frac{y_{C/2} + y_{C/2+1}}{2}$. The exact LAD over $\{y_i\}$ is computed as: $\sum_{i=1}^{C} |y_i - m| = \sum_{i=1}^{C/2} (m - y_i) + \sum_{i=C/2+1}^{C} (y_i - m) = \sum_{i=1}^{C/2} (-y_i) + \sum_{i=C/2+1}^{C} y_i$. If $\{y_i\}$ are grouped according to the bins in histogram H, we can obtain

$$\sum_{i=1}^{C/2}(-y_i) + \sum_{i=C/2+1}^{C} y_i = \underbrace{\sum_{j=b_1}^{j=b_{m-1}} \sum_{y_i \in j} (-y_i)}_{(a)} + \underbrace{\sum_{j=b_{m+1}}^{j=b_\beta} \sum_{y_i \in j} y_i}_{(b)} + \underbrace{\sum_{y_i \in b_m \text{and } i \leq R} (-y_i)}_{(c)} + \underbrace{\sum_{y_i \in b_m \text{and } i > R} y_i}_{(d)}.$$

Item (a) and (b) in above formula can be exactly calculated by the sum element in the corresponding bin of the histogram. Item (c) and (d) can be computed through the R-partial-sum of the 0.5-quantile bin, namely, $-S_p(b_m, R)$ and $b_m.s - S_p(b_m, R)$. Above derivation also applies to the case that C is odd. The lemma is proved.
Likewise, as for TLAD assume that the target values of data instances summarized by histogram H are given as a sorted list $\{y_i\}$ and $C = count(H)$ is

even. The exact Trimmed-LAD for k trimmed dataset ($k = \tau C$) is expressed as $\sum_{i=k+1}^{C/2}(-y_i) + \sum_{i=C/2+1}^{C-k} y_i$. By grouping the individual values into the corresponding bins in the histogram, we can obtain $\sum_{i=k+1}^{C/2}(-y_i) + \sum_{i=C/2+1}^{C-k} y_i =$

$$\underbrace{\sum_{y_i \in b_q \text{and } i > R_1} (-y_i)}_{(a)} + \underbrace{\sum_{j=b_{q+1}}^{j=b_{m-1}} \sum_{y_i \in j} (-y_i)}_{(b)} + \underbrace{\sum_{j=b_{m+1}}^{j=b_{\overline{q}-1}} \sum_{y_i \in j} y_i}_{(c)} + \underbrace{\sum_{y_i \in b_m \text{and } i \leq R} (-y_i)}_{(d)} + \underbrace{\sum_{y_i \in b_m \text{and } i > R} (y_i)}_{(e)} + \underbrace{\sum_{y_i \in b_{\overline{q}} \text{and } i \leq R_2} (y_i)}_{(f)}$$

. Then, item (a), (d), (e) and (f) can be respectively derived by R-partial sums of the corresponding bins and (b) and (c) can be exactly calculated in the histogram.□

Theorem 1. *Given a bin $b = (l, r, c, s)$ of a histogram and R ($R \leq b.c$), if $R = 1$ or $R = b.c$, $\hat{S}_p(b, R)$ provided in Lemma 1 is the exact R-Partial-Sum. Otherwise, the approximation error of R-Partial-Sum of bin b, $S_p(b, R) - \hat{S}_p(b, R)$ is bounded within $[(R - b.c) \cdot (b.r - b.l), b.s - b.c \cdot b.l]$.*

Proof. We first derive the bounds of the exact R-partial sum and the approximate one. Based on Lemma 1, when $R = b.c$, the approximate partial sum is equal to the exact value. For the case of $R < b.c$, $\hat{S}_p(b, R) = R \cdot b.l + R(R-1)\delta$, where $\delta = \frac{(b.s - b.r - b.c \cdot b.l + b.l)}{(b.c-2)(b.c-1)}$. Therefore, we can derive that $R \cdot b.l \leq \hat{S}_p(b, R) \leq R \cdot b.l + b.s - b.c \cdot b.l$. For the exact value $S_p(b, R)$, we have the first bound below: $b.s - S_p(b, R) \leq (b.c - R) \cdot b.r$. It holds because the sum of the remaining values, i.e. $b.s - S_p(b, R)$ is maximized when all the remaining values locate on the right boundary of the bin $b.r$. As a result, we can obtain $b.s - (b.c - R) \cdot b.r \leq S_p(b, R)$. Similarly, the upper bound is derived by $S_p(b, R) \leq b.s - (b.c - R) \cdot b.l$. Now, merging the bounds of $\hat{S}_p(b, R)$ and $S_p(b, R)$ leads to the theorem.