Fighting biases with dynamic boosting

Anna Veronika Dorogush Yandex
Andrey Gulin Yandex
Gleb Gusev Yandex
Nikita Kazeev Yandex
Liudmila Ostroumova Prokhorenkova Yandex
Aleksandr Vorobev Yandex

Abstract

While gradient boosting algorithms are the workhorse of modern industrial machine learning and data science, all current implementations are susceptible to a non-trivial but damaging form of label leakage. It results in a systematic bias in pointwise gradient estimates that lead to reduced accuracy. This paper formally analyzes the issue and presents solutions that produce unbiased pointwise gradient estimates. Experimental results demonstrate that our open-source implementation of gradient boosting that incorporates the proposed algorithm produces state-of-the-art results outperforming popular gradient boosting implementations.

1 Introduction

Gradient boosting is a powerful machine-learning technique that achieves state-of-the-art results in a variety of practical tasks. For a number of years, it has remained the primary method for learning problems with heterogeneous features, noisy data, and complex dependencies: web search, recommendation systems, weather forecasting, and many others Caruana and Niculescu-Mizil [2006], Roe et al. [2005], Wu et al. [2010], Zhang and Haghani [2015]. It is backed by strong theoretical results that explain how strong predictors can be built by iteratively combining weaker models (typically decision trees) via a greedy procedure that corresponds to gradient descent in function space.

In this paper, we identify and solve the problem of bias in pointwise gradient estimates that is present in all classical boosting algorithms and modern industrial implementations. Gradients used at each step are estimated using the same data points that produced the model built so far. This causes an undesirable dependency between the data used for the gradient estimation and the model approximation. One simple consequence is that the residuals estimated on the data points used for training tend to have smaller absolute values than those for unseen data. We show how this estimation bias affects the quality of learning (Sec. 2), and propose a principled way to mitigate the problem.

Formal analysis of the gradient bias problem allows us to propose a *dynamic boosting* approach that avoids the estimation bias at a minimal cost of the variance of the gradient estimation (Sec. 3). We prove, under reasonable assumptions, that this approach ensures unbiased residuals for regression tasks (Sec. 4). The idea of biased gradients and residuals was discussed in previous literature, and heuristic techniques have been proposed, such as iterated bagging Breiman [1996] and subsampling the dataset at each step of gradient boosting Friedman [2002]. However, the problem of estimation bias was still poorly understood and the proposed methods did not guarantee the unbiasedness.

While the dynamic boosting algorihtm is computationally inefficient, we propose several algorithmic tricks to construct feasible implementations of the method (Sec 5). The resulting implementation of dynamic boosting outperforms XGBoost Chen and Guestrin [2016] and LightGBM[1], existing state-of-the-art open-source implementations of gradient boosted decision trees (GBDTs), on a diverse set

[1] `https://github.com/Microsoft/LightGBM`

of classification and regression tasks (Sec. 6). Our algorithm is to be released in open-source, and the link will be provided in the camera–ready version of this paper.

1.1 Background on gradient boosting

Assume we are given a dataset of observations $\mathcal{D} = \{(\mathbf{X}_k, Y_k)\}_{k=1..n}$, where $\mathbf{X}_k \in \mathbb{R}^m$ is a random vector of m *features* and $Y_k \in \mathbb{R}$ is a *target*. Let the observations $(\mathbf{X}_k, Y_k)$ be independent and identically distributed according to some unknown distribution $P(\mathbf{x}, y)$. We consider a general setting of learning task where the goal is to train a function $F\colon \mathbb{R}^m \to \mathbb{R}$ which minimizes the expected loss $\mathcal{L}(F) := \mathbb{E}L(Y, F(\mathbf{X}))$, where $L(\cdot, \cdot)$ is a smooth loss function and $(\mathbf{X}, Y)$ is a *test example* sampled from $P(\mathbf{x}, y)$ independently of any other observations. A gradient boosting procedure Friedman [2001] builds iteratively a sequence of *approximations* $F^t\colon \mathbb{R}^m \to \mathbb{R}$, $t = 0, 1, \ldots$ in a greedy fashion. Namely, F^t is obtained from the previous approximation F^{t-1} in an additive manner:

$$F^t = F^{t-1} + \alpha^t h^t, \tag{1}$$

where α^t is a *step size* and function $h^t\colon \mathbb{R}^m \to \mathbb{R}$ (a *base learner*) is chosen from a family of functions H^t Cortes et al. [2014] in order to minimize expected predictive loss:

$$h^t = \operatorname{argmin}_{h \in H^t} \mathcal{L}(F^{t-1} + h) = \operatorname{argmin}_{h \in H^t} \mathbb{E}L(Y, F^{t-1}(\mathbf{X}) + h(\mathbf{X})). \tag{2}$$

This minimization problem is usually approached by the *Newton method* using a second–order approximation of $\mathcal{L}(F^{t-1} + h^t)$ at F^{t-1} or is substituted with a *(negative) gradient step*. Both methods are kinds of functional gradient descent Friedman et al. [2000], Mason et al. [2000]. In particular, the gradient step h^t is chosen in such a way that $h^t(\mathbf{X})$ approximates -$g^t(\mathbf{X}, Y)$, where $g^t(\mathbf{x}, y) := \frac{\partial L(y,s)}{\partial s}\big|_{s=F^{t-1}(\mathbf{x})}$. Usually, the least-squares approximation is used:

$$h^t = \operatorname{argmin}_{h \in H^t} \mathbb{E}\left(-g^t(\mathbf{X}, Y) - h(\mathbf{X})\right)^2. \tag{3}$$

In practical tasks, the expectation in (3) is unknown and is usually approximated using the same dataset $\mathcal{D}$ or its (bootstrap) subsample $\mathcal{D}' = \{(\mathbf{X}'_k, Y'_k)\}_{k=1..n'}$:

$$h^t = \operatorname{argmin}_{h \in H^t} \frac{1}{n'} \sum_{k=1}^{n'} \left(-g^t(\mathbf{X}'_k, Y'_k) - h(\mathbf{X}'_k)\right)^2. \tag{4}$$

In this paper, we show that the approximation (4) is strongly biased and may result in overfitting. Further in this section, we study the source of estimation bias and describe it formally. In the next section, we show how this estimation bias affects the quality of learning. In Sec. 3, we propose a general framework for solving the gradient bias issue. In Sec 5, we apply this framework to the special case of the gradient boosting with H^t being a family of decision trees [Friedman, 2001, Section 4.3]. In this case, a base learner h can be written as $h(x) = \sum_{j=1}^{J} b_j \mathbb{1}_{\{x \in R_j\}}$, where R_j are disjoint regions covering $\mathbb{R}^m$. Each region corresponds to a leaf (terminal node) of a tree. The parameters of h are the splitting points at the nodes (they define regions) and the values b_j. The splitting points are usually chosen in a top-down greedy manner. In the case of regression task, both splitting points and the values of b_j are usually chosen using the least-squares splitting criterion Breiman et al. [1984], Friedman et al. [2000].

1.2 Estimation Bias of Pointwise Gradient

The approximation (4) is an unbiased estimate of (3), when the dataset $\mathcal{D}'$ is independent from the dataset $\mathcal{D}$ used for training the approximation F^{t-1}. Moreover, in this case we have $\mathbb{E}(L(Y'_k, F^{t-1}(\mathbf{X}'_k) \mid \mathbf{X}'_k = \mathbf{x}) = \mathcal{L}(F^{t-1}, \mathbf{x})$ and $\mathbb{E}\mathcal{L}(F, \mathbf{X}) = \mathcal{L}(F)$, where $\mathcal{L}(F, \mathbf{x}) := \mathbb{E}L(Y, F(\mathbf{X}) \mid \mathbf{X} = \mathbf{x})$ is the *local loss* at point $\mathbf{x}$. That is, the loss at the observation $(\mathbf{X}'_k, Y'_k)$ is an unbiased estimate of the corresponding local loss.

We emphasize that things dramatically change when we condition on some dependence between $\mathcal{D}'$ and $\mathcal{D}$. Indeed, at each step t, a boosting algorithm aims to reduce the current errors on training examples. Therefore, one expects that $\mathbb{E}(L(Y'_k, F^{t-1}(\mathbf{X}'_k) \mid \mathbf{X}'_k = \mathbf{x}) < \mathcal{L}(F^{t-1}, \mathbf{x})$, since the target values of these examples were directly used at the previous iterations of the algorithm. We

further study the particular case of a usual regression problem with $L(y, s) = (y - s)^2$ for simplicity. In this case, the problem (3) becomes:

$$h^t = \text{argmin}_{h \in H^t} \mathbb{E} \left(r^{t-1}(\mathbf{X}, Y) - \frac{1}{2} h(\mathbf{X}) \right)^2, \tag{5}$$

where $r^{t-1}(\mathbf{x}, y) := y - F^{t-1}(\mathbf{x})$ is the residual function, and Equation (4) is equivalent to

$$h^t = \text{argmin}_{h \in H^t} \frac{1}{n'} \sum_{k=1}^{n'} \left(r^{t-1}(X'_k, Y'_k) - \frac{1}{2} h(\mathbf{X}'_k) \right)^2. \tag{6}$$

In the naïve approach with $\mathcal{D}' = \mathcal{D}$, Equation (6) does not provide an unbiased estimate, since the equation $\mathbb{E}(r^{t-1}(\mathbf{X}_k, Y_k) \mid \mathbf{X}_k = \mathbf{x}) = \mathbb{E}(r^{t-1}(\mathbf{X}, Y) \mid \mathbf{X} = \mathbf{x})$ does not hold in general. In fact, the difference between the right– and the left–hand sides is

$$B_t(\mathbf{x}) := \mathbb{E}(F^{t-1}(\mathbf{X}) \mid \mathbf{X} = \mathbf{x}) - \mathbb{E}(F^{t-1}(\mathbf{X}) \mid \mathbf{X}_k = \mathbf{X} = \mathbf{x}), \tag{7}$$

what is the (expected) change in the value of the function F^{t-1} at some point $\mathbf{x}$ caused by substitution of an observation $(\mathbf{X}_k, Y_k)$ at the fixed point $\mathbf{X}_k = \mathbf{x}$ by an arbitrary observation.

The *gradient estimation bias* $B_t(\mathbf{x})$ is the main focus of the study.[2] The intuition that non-zero $B_t(\mathbf{x})$ causes overfitting motivated different techniques like stacked generalization Wolpert [1992], however, a principled solution has not been proposed to the best of our knowledge. To motivate our approach, we start with an empirical study in Sec. 2 that reveals how the bias of the gradient estimation increases the error of the trained model.

2 Empirical Impact of Gradient Estimation Bias

Similarly to Breiman [2001], Friedman [1991], we consider synthetic datasets generated from the following distribution. Let each datapoint $\mathbf{X} = (X_1, \ldots, X_m)$ be a vector of m i.i.d. Bernoulli random variables X_i with parameter $p = 1/2$ and with label $Y = \sum_{i=1}^{m} X_i + \varepsilon$, where $\varepsilon \sim \mathcal{N}(0, \sigma^2)$ is random noise.

We consider base learners to be decision stumps and compare the following boosting algorithms: (i) Plain: standard GBDT (ii) Resample: GBDT which at each iteration generates a new dataset $\mathcal{D}$ of size n from the distribution $P(\mathbf{x}, y)$, as discussed in Sec. 3, (iii) Strict-Plain GBDT, described below in Sec. 5, which aims to mitigate the gradient estimation bias.

In the following experiments, we take $|\mathcal{D}| = 100, m = 20, \sigma = 2, I = 600$. At each step t we estimate the squared residual estimation bias integrated over $\mathbf{X}$:

$$\widehat{B}_t := \mathbb{E}(B_t(\mathbf{X})^2), \tag{8}$$

where B_t is defined by Equation (7). The numerical computation of $\widehat{B}_t$ is described in the supplementary materials. Note that for Resample we have $\widehat{B}_t = 0$ by construction. On Figure 1 we see that Plain GBDT has the largest estimation bias, and its MSE does not converge to the MSE of Resample, illustrating that bias results in suboptimal model estimation. In contrast, Strict-Plain approximation has lower bias (discussed in Sec. 5) and outperforms Plain GBDT.

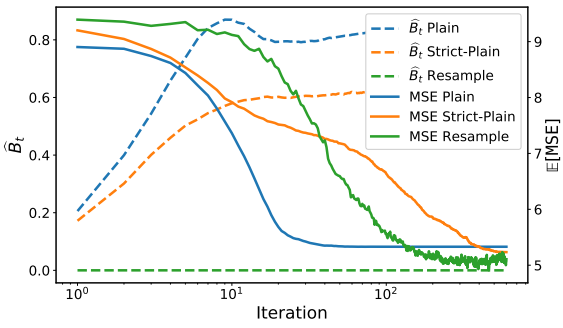

Figure 1: Analysis of $\widehat{B}_t$

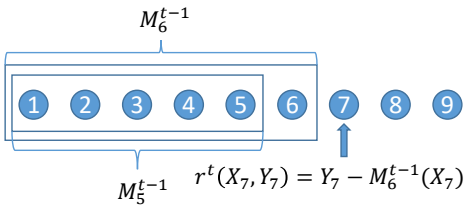

Figure 2: Ordered dynamic boosting

[2] Note that the bias of the gradient (residual) estimation has no direct relationship to the notion of bias-variance trade-off from statistical learning theory James et al. [2013].

3 Fighting Gradient Bias

Let us develop a boosting algorithm which does not suffer from the problem described in Sec. 1.2. We consider the case of a regression task where, at each step, the base learner $h^t(\mathbf{X})$ approximates the residual function $r^{t-1}(\mathbf{X}, Y)$ (see Equation (5)). According to intuition from our empirical results in Sec. 2 (and our theoretical analysis in the next section), it is highly desirable to use *unbiased residuals*.

Assuming access to an unlimited amount of training data, we can easily construct such an algorithm. At each step of boosting, we sample a new dataset $\mathcal{D}_t$ and obtain unbiased residuals by applying the current model to new training samples, so $B_t(\mathbf{x}) = 0$ for any $\mathbf{x}$. In practice, however, labeled data is typically limited, and this solution leads to an undesirable trade-off between the size of the datasets $\mathcal{D}_t$ and biasedness of the residual estimation.

Let us suppose that the dataset $\mathcal{D} = \{(\mathbf{X}_k, Y_k)\}_{k=1}^n$ is limited, but computational resources and memory are infinite. Assume that we want to make I iterations of boosting. To make the residual $r^{I-1}(\mathbf{X}_k, Y_k)$ unbiased w.r.t. the model F^{I-1}, we need to have F^{I-1} trained without the observation $\mathbf{X}_k$. Since we need unbiased residuals for all training examples, no observations may be used for training F^{I-1}, which at first glance makes the training process impossible. We consider the following trick to deal with this problem. For each sample $(\mathbf{X}_k, Y_k)$, we train a separate model M_k^{I-1} on samples $[1, n] \setminus k$ for the previous iterations in order (i) to make the residual $r^{I-1}(\mathbf{X}_k, Y_k) = Y_k - M_k^{I-1}(\mathbf{X}_k)$ on this sample unbiased, and (ii) to use as many samples for model training as possible. Note that models $M_1^{I-1}, \ldots, M_n^{I-1}$ have very similar training datasets and could be interpreted as approximations of a virtual model F^{I-1}. Similarly, at iteration $I-1$, to update each model M_k^{I-1} on the basis of unbiased residuals, we use residuals $r_k^{I-2}(\mathbf{X}_l, Y_l) = Y_l - M_{k,l}^{I-2}$, where $M_{k,l}^{I-2}$ is trained on samples $[1, n] \setminus \{k, l\}$, and so on. As a result, we obtain Algorithm 1, which we call *exponential dynamic boosting*.

Algorithm 1: Exponential dynamic boosting

input : $\{(\mathbf{X}_k, Y_k)\}_{k=1}^n$, I;

$M_S^0 \leftarrow 0$ for all $S \subset [1, n], |S| \leq I$;
for $iter \leftarrow 1$ **to** I **do**
  foreach S *s.t.* $|S| \leq I - iter$ **do**
    foreach $i \in [1, n] \backslash S$ **do**
      $r_i \leftarrow Y_i - M_{S \cup i}^{iter-1}(i)$;
    $M \leftarrow LearnModel((\mathbf{X}_i, r_i)$ for $i \in [1, n] \backslash S$;
    $M_S^{iter} \leftarrow M_S^{iter-1} + M$;
return $M_\emptyset^I$

Algorithm 2: Ordered dynamic boosting

input : $\{(\mathbf{X}_k, Y_k)\}_{k=1}^n$, I;

$M_i \leftarrow 0$ for $i = 1..n$;
for $iter \leftarrow 1$ **to** I **do**
  for $i \leftarrow 1$ **to** n **do**
    $r_i \leftarrow y_i - M_{i-1}(i)$;
  for $i \leftarrow 1$ **to** n **do**
    $M \leftarrow LearnModel((\mathbf{X}_j, r_j)$ for $j = 1..i-1)$;
    $M_i \leftarrow M_i + M$;
return M_n

This algorithm allows to iteratively construct a model having unbiased residuals r_i at each iteration. However, such an algorithm requires huge amount of resources to learn $\Theta\left(n^I / I!\right)$ models M_S^{iter}. At the same time, it can be significantly simplified by introducing the *order* on the dataset. Namely, let us assume that (X_i, Y_i) are randomly ordered. We maintain only n models $M_1, \ldots, M_n$ such that the model M_i is learned using only the first i samples. At each step, in order to obtain the residual for j-th sample, we use the model M_{j-1} (see Figure 2). The resulting Algorithm 2 is called *ordered dynamic boosting* below. Unlike Algorithm 1, this procedure has polynomial complexity. However, it is still not feasible in most practical tasks, since it iteratively builds n models. What if we are restricted to only one base learner per iteration? It is clear from the above reasoning that, in this case, we cannot avoid biased residuals. However, we reduce this bias by modifying the plain gradient boosting algorithm with decision trees as base learners (GBDT) in Sec. 5.

4 Theoretical guarantees on unbiasedness

We prove below that removing bias in the residuals guarantees a model that is unbiased with respect to the following "ideal" function.

Definition. An *oracle function* is a function $\widehat{F}^I = \sum_{t=1}^I \alpha^t \widehat{h}^t$ built by the gradient boosting procedure, where $\widehat{h}^t$ is an exact (deterministic) solution of Equation (5).

This oracle function is not achievable in practice, since the expectation from the left–hand side of (3) is unknown.

Definition. We call a sequence of families H^t *convenient*, if the following conditions are satisfied:

1. Family H^t is a linear space;
2. At each step t, there exists a unique minimizer h_t of Equation (6) based on the dataset $\mathcal{D}$;
3. We have $\mathbb{E}(h^t(\mathbf{x})) = \widehat{h}^t(\mathbf{x})$ for any $\mathbf{x} \in \mathbb{R}^m$, where

$$\widehat{h}^t = \operatorname{argmin}_{h \in H^t} \mathbb{E}\left(Y - \frac{1}{2}h(\mathbf{X})\right)^2, \quad h^t = \operatorname{argmin}_{h \in H^t} \frac{1}{n}\sum_{k=1}^n \left(Y_k - \frac{1}{2}h(\mathbf{X}_k)\right)^2$$

and $(\mathbf{X}, Y)$, $(\mathbf{X}_k, Y_k)$ are i.i.d.

Theorem 1 *Assume H^t is convenient for any $t \leq I$. Then Algorithm 2 provides an unbiased approximation F_n^I in the sense that $\mathbb{E}(F_n^I(\mathbf{x})) = \mathbb{E}(\widehat{F}^I(\mathbf{x}))$ for any $\mathbf{x} \in \mathbb{R}^m$, where $\widehat{F}^I$ is the oracle function.*

The proof of this theorem is available in the supplementary materials. For example, let H_t be the set of decision trees $\{h_{\{b_j\}} := \sum_{j=1}^J b_j \mathbb{1}_{\{x \in R_{t,j}\}}\}$ with a fixed structure $\mathcal{R}_t = \{R_{t,j}\}$. The sequence H_t is convenient, if any leaf $R_{t,j}$ contains at least one observation from $\mathcal{D}$. Unfortunately, we cannot drop the condition that the structure is given in advance and is not trained in the boosting procedure. We believe, however, that Theorem 1 explains why unbiased residuals are important and motivate our ordering principle proposed in Sec. 3.

5 Dynamic boosting decision trees

While Algorithm 2 above provided strong guarantees for reducing residual bias (Sec. 4), it is inefficient due to the need of maintaining n different ensemble models. In this section, we construct several modifications of GBDT based on the same idea of ordering as Algorithm 2. Using trees as base learners, on the one hand, is the state-of-the-art choice Caruana and Niculescu-Mizil [2006], Roe et al. [2005], Wu et al. [2010], and, on the other hand, allows us to apply the ordering principle while learning only one tree at each iteration.

Each principal modification of the algorithm is specified by the two hyperparameters, $SplitMode$ and $LeafMode$. They define how the algorithm averages gradients (corresponding to residuals in the above-described regression problem) for constructing tree structure $\{R_j\}$ (lines 7 and 9 of Function BuildTree) and for setting the values $\{b_j\}$ in leaves (Function CalcLeafValue), respectively. Each parameter could be set to $Plain$ mode corresponding to the classic GBDT algorithm, $Strict$ or $Soft$ modes.

In the last two modes, called strict/soft ordering, we generate $2s$ random permutations of our training dataset (permutations $\{\sigma_i\}_{i=1}^s$ are used for choosing splits and $\{\sigma_i\}_{i=s+1}^{2s}$ for setting values in leaves). We use several permutations to enhance the robustness of the algorithm: at each iteration, we sample a random permutation and obtain gradients on the basis of it. Namely, in strict ordering mode ($Strict$), the gradient $grad_r(i)$ on sample i is calculated on the basis of the prediction $S_r(i)$ learned using the preceding samples (see line 8 of Function UpdateModel), or, more precisely, using their gradients $grad_r(i)$ obtained under the same ordering principle.

The difference of the soft ordering mode ($Soft$) is that the gradient $grad_{r,\sigma_r(i)}(p)$ of an example p used for training the prediction $S_{r,\sigma_r(i)}(i)$ (which is used for choosing splits or setting values in leaves) for sample i, $\sigma_r(p) < \sigma_r(i)$, is obtained on the basis of samples j with $\sigma_r(j) < \sigma_r(i)$, some of which are not preceding for the sample p. Thus, in this mode, gradients are stronger biased, but have lower variance, since the corresponding predictions are trained on the larger amount of examples. Note that this mode maintains a prediction model $S_{r,i}$ for each pair of index i, $1 \leq i \leq n$, and

Algorithm 3: Ordered dynamic boosting

input : $\{(\mathbf{X}_k, Y_k)\}_{k=1}^n$, I, α, L, s, $SplitMode$, $LeafMode$

$\sigma_i \leftarrow$ random permutation of $[1, n]$ for $i = 1..2s$;
$S_{r,j}(i) \leftarrow 0$ for $r = 1..2s$, $i = 1..n$, $j = 1..n$;
for *iter* $\leftarrow 1$ **to** I **do**
  $grad \leftarrow CalcGradient(L, S, Y)$;
  $r \leftarrow random(1, s)$;
  $T \leftarrow BuildTree(SplitMode, grad, \sigma_r, \{\mathbf{X}_k\}_{k=1}^n)$;
  foreach *leaf* R_j^{iter} *in* T **do**
    $b_j^{iter} \leftarrow CalcLeafValue(LeafMode, grad, \{\mathbf{X}_k\}_{k=1}^n, R_j^{iter})$;
  $\{S\}_{r=1}^s \leftarrow UM(SplitMode, T, \{S_r\}_{r=1}^s, grad, \{\sigma_r\}_{r=1}^s)$;
  $\{S\}_{r=s+1}^{2s} \leftarrow UM(LeafMode, T, \{S_r\}_{r=s+1}^{2s}, grad, \{\sigma_r\}_{r=s+1}^{2s})$;
return $F(\mathbf{x}) = \sum_{i=1}^I \sum_j \alpha\, b_j^i \mathbb{1}_{\{\mathbf{x} \in R_j^i\}}$;

Function UpdateModel(UM)

input : $Mode$, T, S, $grad$, σ_i $for\ i = 1..s$

foreach *leaf* R_j *in* T **do**
  foreach *i s.t.* $\mathbf{X}_i \in R_j$ **do**
    if $Mode == Plain$ **then**
      $S_{1,1}(i) \leftarrow S_{1,1}(i) - \alpha\,\text{avg}(grad_{1,1}(l)\ for\ \mathbf{X}_l \in R_j)$;
    else
      for $r \leftarrow 1$ **to** s **do**
        if $Mode == Strict$ **then**
          $S_{r,1}(i) \leftarrow S_{r,1}(i) - \alpha\,\text{avg}(grad_{r,1}(p)\ for\ \mathbf{X}_p \in R_j\ s.t.\ \sigma_r(p) < \sigma_r(i))$;
        if $Mode == Soft$ **then**
          for $l \leftarrow 1$ **to** n **do**
            $S_{r,l}(i) \leftarrow S_{r,l}(i) - \alpha\,\text{avg}(grad_{r,l}(p)\ for\ \mathbf{X}_p \in R_j\ s.t.\ \sigma_r(p) < l)$;
return S

Function SetGradient

input : $Mode$, $grad$, σ_r

if $Mode == Plain$ **then**
  $G(i) \leftarrow grad_{1,1}(i)\ for\ i = 1..n$;
if $Mode == Strict$ **then**
  $G(i) \leftarrow grad_{r,1}(i)\ for\ i = 1..n$;
if $Mode == Soft$ **then**
  $G(i) \leftarrow grad_{r,\sigma_r(i)}(i)\ for\ i = 1..n$;
return G

Function BuildTree

input : $Mode$, $grad$, σ_r, $\{\mathbf{X}_k\}_{k=1}^n$

$G = SetGradient(Mode, grad, \sigma_r)$;
foreach *step of top-down procedure* **do**
  foreach *candidate split* T' **do**
    foreach *leaf* R_j *in* T' **do**
      foreach $\mathbf{X}_i \in R_j$ **do**
        if $Mode == Plain$ **then**
          $\Delta(i) \leftarrow \text{avg}(G(l)\ for\ \mathbf{X}_l \in R_j)$;
        else
          $\Delta(i) \leftarrow \text{avg}(G(l)\ for\ \mathbf{X}_l \in R_j, \sigma_r(l) < \sigma_r(i))$;
    $loss(T') \leftarrow ||\Delta - G||_2$
  $T \leftarrow \text{argmin}_{T'}(loss(T'))$
return T

Function CalcLeafValue

input : $Mode$, $grad$, $\{\mathbf{X}_k\}_{k=1}^n$, R

if $Mode == Plain$ **then**
  $b \leftarrow \text{avg}(grad_{1,1}(i)\ for\ \mathbf{X}_i \in R)$;
if $Mode == Strict$ **then**
  $b \leftarrow \text{avg}(grad_{r,1}(i)\ for\ \mathbf{X}_i \in R,\ r = (s+1)..2s))$;
if $Mode == Soft$ **then**
  $b \leftarrow \text{avg}(grad_{r,\sigma_r(i)}(i)\ for\ \mathbf{X}_i \in R,\ r = (s+1)..2s))$;
return b

permutation σ_r. For uniformity of notations, we denote predictions and corresponding gradients for other modes by substituting indices of absent dimensions by 1 in the pseudocode (like $S_{r,1}(i)$ or $grad_{1,1}(i)$).

Thus, we expect that gradients used in $Strict$ mode are the lowest-biased but have the highest variance, whereas gradients in $Plain$ mode might be the highest-biased and have the lowest variance. According to our empirical results in Figure 1, the modification *Strict–Plain* of the ordering boosting algorithm learned a better model than *Plain–Plain* one (called $Plain$ on Figure 1). Therefore, we believe that the plain GBDT does not provide the optimal trade-off between bias and variance of gradients. Since we have no theoretical arguments for why any mode in any of two components of Algorithm 3 should provide a better trade-off, we experiment with all nine modifications in Sec. 6.

Table 1: Comparison of modifications

	AUC for classification tasks		MSE for regression tasks	
	Springleaf*	Adult	Liberty	Abalone
Plain-Plain	0.7549	0.9264	3.6802	2.3092
Plain-Soft	0.7515	0.9258	3.6987	2.3003
Plain-Strict	0.7546	0.9265	**3.6748**	2.3020
Soft-Plain	**0.7560**	**0.9285**	3.6790	**2.2937**
Soft-Soft	0.7363	0.9250	3.6774	2.3016
Soft-Strict	0.7516	0.9258	3.6784	2.3055
Strict-Plain	0.7542	0.9267	3.6854	2.3004
Strict-Soft	0.7546	0.9270	3.6784	2.3053
Strict-Strict	0.7517	0.9265	3.6795	2.3069

6 Experiments

Data We conduct experiments on several public datasets of various sizes for both classification and regression learning tasks. Datasets Adult and Abalone are from the UCI Machine Learning Repository[3], while other datasets are from the Kaggle machine learning competition website: Allstate Claim Prediction Challenge[4], Springleaf Marketing Response[5], and Liberty Mutual Group: Property Inspection Prediction[6].

Most of the datasets are randomly split into train and test set, the latter consisting of 20% samples. The only exception is Allstate: this dataset is naturally ordered, so the samples from 2005 and 2006 are used as the train set, and from 2007 as the test set. Also, in order to treat categorical features in a unified manner, for all datasets containing such features we randomly and uniformly split the train dataset into holdout and train folds w.r.t. categorical features. Using estimates from the holdout fold, for classification tasks we replace categorical features by the conditional probability estimates, as described in Cestnik et al. [1990], Micci-Barreca [2001]. Similarly, for regression tasks, we replace categorical features by their expected target values estimated on the holdout dataset.

Also, to speed up the computation, in some cases we use a reduced version of Springleaf dataset: train fold of Springleaf* has 4% of the whole dataset, while holdout and test folds are kept unchanged.

Compared algorithms We compare all modifications of dynamic boosting described in Sec. 5. To implement the algorithms, we use oblivious decision trees, where the same splitting criterion is used across an entire level of the tree Kohavi and Li [1995], Langley and Sage [1994]. Such trees are balanced, less prone to overfitting, and allow speeding up prediction significantly at testing time. Gradient boosted oblivious trees were successfully used in various learning tasks Ferov and Modrỳ [2016], Gulin et al. [2011]. $L2$ regularization was used with all algorithms, as well as feature subsampling. Three training permutations were used where applicable ($s = 3$ in Algorithm 3). For all algorithms optimal hyperparameters where chosen using the hyperopt library[7] and 5-fold cross-validation.

Analysis of dynamic boosting First, let us compare the algorithms proposed in Sec. 5. In Table 1, we present AUC (for classification tasks) and MSE (for regression tasks) for all 9 modifications of Algorithm 3. We noticed that *Soft–Plain* modification shows the best performance on almost all the datasets (except one). Also, in all cases this modification performs significantly (p-value<0.05 by paired t-test) better than *Plain–Plain* modification which is an implementation of standard GBDT.

[3] `https://archive.ics.uci.edu/ml/datasets.html`

[4] `https://www.kaggle.com/c/ClaimPredictionChallenge/data`

[5] `https://www.kaggle.com/c/springleaf-marketing-response/data`

[6] `https://www.kaggle.com/c/liberty-mutual-group-property-inspection-prediction/data`

[7] `https://github.com/hyperopt/hyperopt`

Table 2: Comparison with baselines

	AUC for classification tasks			MSE for regression tasks	
	Allstate	Springleaf	Adult	Liberty	Abalone
XGBoost	0.6038	0.7738	0.9111	3.6988	2.3044
LightGBM	0.6006	0.7743	0.9121	3.6909	**2.2926**
Soft-Plain	**0.6138**	**0.7791**	**0.9244**	**3.6790**	2.2937

Comparison with baselines We compare our algorithm with two open-source systems: XGBoost Chen and Guestrin [2016] and LightGBM[8]. Both systems perform tree boosting and are widely used in machine learning community. For LightGBM we took the parameters used in [9], the parameters for XGBoost were chosen according to [Chen and Guestrin, 2016].

The results are presented in Table 2. Note that the modification *Soft–Plain* of the ordered dynamic boosting algorithm significantly (p-value<0.05 by paired t-test) outperforms the baselines on 4 out of 5 datasets, whereas the difference from LightGBM on Abalone is not significant. Thus, this modification provides the best trade-off between bias and variance of gradients in terms of performance of the resulting model.

7 Related work

The problem of overfitting caused by correlation between the noise in the data and the outputs of approximations was considered previously in papers on boosting Breiman [2001], Friedman [2002]. Based on the out-of-bag estimation first proposed in his paper Breiman [1996], Breiman proposed *iterated bagging* Breiman [2001] which simultaneously constructs K models F_i, $i = 1, \ldots, K$, associated with K independently bootstrapped subsamples $\mathcal{D}_i$. At t-th step of the process, approximations F_i^t are grown from their predecessors F_i^{t-1} as follows. The current estimate M_j^t at example j is obtained as the average of the outputs of all models F_k^{t-1} such that $j \notin \mathcal{D}_k$. Base learner h_i^t is build as a predictor of the residuals $r_j^t := Y_j - M_j^t$ (targets minus current estimates) on $\mathcal{D}_i$. Finally, the models are updated: $F_i^t := F_i^{t-1} + h_i^t$.

Unfortunately, the residuals r_j^t used in this procedure are not unbiased in the sense of Sec. 1.2 (see Equation (7)), because each model F_i^t depends on each observation $(\mathbf{X}_j, Y_j)$ by construction. Indeed, although h_k^t does not use Y_j directly, if $j \notin \mathcal{D}_k$, it still uses $M_{j'}^{t-1}$ for $j' \in \mathcal{D}_k$, which, in turn, can depend on $(\mathbf{X}_j, Y_j)$. The same is applicable to the idea of Friedman Friedman [2002], where subsampling of the dataset at each iteration was proposed.

Acknowledgments

We would like to thank Mikhail Bilenko, Stanislav Kirillov, Victor Omelyanenko, and Pavel Serdyukov.

8 Conclusions

In this paper, we identify the problem of biased pointwise gradient estimates that results in overfitting and corresponding accuracy deterioration for stochastic gradient boosting. Following formal analysis, we propose a general solution that removes the estimation bias, which is computationally infeasible, and a practical approximation of this solution. Empirical results demonstrate that our approach leads to state-of-the-art results on common benchmarks that outperform leading GBDT packages.

References

L. Breiman. Out-of-bag estimation, 1996.

[8] `https://github.com/Microsoft/LightGBM`

[9] `https://github.com/Microsoft/LightGBM/wiki/Experiments`

L. Breiman. Using iterated bagging to debias regressions. *Machine Learning*, 45(3):261–277, 2001.

L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen. *Classification and regression trees*. CRC press, 1984.

R. Caruana and A. Niculescu-Mizil. An empirical comparison of supervised learning algorithms. In *Proceedings of the 23rd international conference on Machine learning*, pages 161–168. ACM, 2006.

B. Cestnik et al. Estimating probabilities: a crucial task in machine learning. In *ECAI*, volume 90, pages 147–149, 1990.

T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794. ACM, 2016.

C. Cortes, M. Mohri, and U. Syed. Deep boosting. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1179–1187, 2014.

M. Ferov and M. Modrỳ. Enhancing lambdamart using oblivious trees. *arXiv preprint arXiv:1609.05610*, 2016.

J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: a statistical view of boosting. *The annals of statistics*, 28(2):337–407, 2000.

J. H. Friedman. Multivariate adaptive regression splines. *The annals of statistics*, pages 1–67, 1991.

J. H. Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

J. H. Friedman. Stochastic gradient boosting. *Computational Statistics & Data Analysis*, 38(4): 367–378, 2002.

A. Gulin, I. Kuralenok, and D. Pavlov. Winning the transfer learning track of yahoo!'s learning to rank challenge with yetirank. In *Yahoo! Learning to Rank Challenge*, pages 63–76, 2011.

G. James, D. Witten, T. Hastie, and R. Tibshirani. *An introduction to statistical learning*, volume 6. Springer, 2013.

R. Kohavi and C.-H. Li. Oblivious decision trees, graphs, and top-down pruning. In *IJCAI*, pages 1071–1079. Citeseer, 1995.

P. Langley and S. Sage. Oblivious decision trees and abstract cases. In *Working notes of the AAAI-94 workshop on case-based reasoning*, pages 113–117. Seattle, WA, 1994.

L. Mason, J. Baxter, P. L. Bartlett, and M. R. Frean. Boosting algorithms as gradient descent. In *Advances in neural information processing systems*, pages 512–518, 2000.

D. Micci-Barreca. A preprocessing scheme for high-cardinality categorical attributes in classification and prediction problems. *ACM SIGKDD Explorations Newsletter*, 3(1):27–32, 2001.

B. P. Roe, H.-J. Yang, J. Zhu, Y. Liu, I. Stancu, and G. McGregor. Boosted decision trees as an alternative to artificial neural networks for particle identification. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 543(2):577–584, 2005.

D. H. Wolpert. Stacked generalization. *Neural networks*, 5(2):241–259, 1992.

Q. Wu, C. J. Burges, K. M. Svore, and J. Gao. Adapting boosting for information retrieval measures. *Information Retrieval*, 13(3):254–270, 2010.

Y. Zhang and A. Haghani. A gradient boosting method to improve travel time prediction. *Transportation Research Part C: Emerging Technologies*, 58:308–324, 2015.

Supplementary materials

The squared bias of the residual estimation

We estimate $\widehat{B}_t$ defined in (8) using bootstrapping in a way similar to the ideas of out-of-bag estimation Breiman [1996]. Namely, we build S bootstrap-samples $\mathcal{D}_1, \ldots, \mathcal{D}_S$ from $\mathcal{D}$ and apply boosting to each of these datasets. We consider S models $F_1^{t-1}, \ldots, F_S^{t-1}$ obtained after $t-1$ steps on these datasets correspondingly. For each $j \in [1, n]$, we use estimations

$$\mathbb{E}(F^{t-1}(\mathbf{X}) \mid \mathbf{X} = \mathbf{x}_j) \approx \frac{1}{S - |L_j|} \sum_{s \notin L_j} F_s^{t-1}(\mathbf{x}_j),$$

$$\mathbb{E}(F^{t-1}(\mathbf{X}_k) \mid \mathbf{X}_k = \mathbf{x}_j) \approx \frac{1}{\sum_s m_{j,s}} \sum_s m_{j,s} F_s^{t-1}(\mathbf{x}_j),$$

4where $\mathbf{x}_j$ is the value of $\mathbf{X}_j$, $m_{j,s}$ is the multiplicity of $(\mathbf{X}_j, Y_j)$ in $\mathcal{D}_s$, and $L_j := \{s \mid m_{j,s} > 0\}$. The estimation of $\widehat{B}_t$ is then obtained as the average of $B_t(\mathbf{x})$ over values of $\mathbf{X}$ observed in $\mathcal{D}$:

$$\widehat{B}_t \approx \frac{1}{n} \sum_{j=1}^{n} \left(\frac{1}{S - |L_j|} \sum_{s \notin L_j} F_s^{t-1}(\mathbf{x}_j) - \frac{1}{\sum_s m_{j,s}} \sum_{s \in L_j} m_{j,s} F_s^{t-1}(\mathbf{x}_j) \right)^2 .$$

Proof of Theorem 1

We perform induction on I. If $I = 1$, then the theorem reduces to the third condition of convenience. Assume that the theorem is valid for $I = t-1$ and prove that $\mathbb{E}(F_n^t(\mathbf{x})) - \widehat{F}^t(\mathbf{x}) = 0$. The first part is equal to $\mathbb{E}(F_n^{t-1}(\mathbf{x})) + \alpha^t \mathbb{E}(h^t(\mathbf{x}))$ and the second one is $\widehat{F}^{t-1}(\mathbf{x}) + \alpha^t \widehat{h}^t(\mathbf{x})$. Since, by induction, $\mathbb{E}(F_n^{t-1}(\mathbf{x})) = \widehat{F}^{t-1}(\mathbf{x})$, we have

$$\mathbb{E}(F_n^t(\mathbf{x})) - \widehat{F}^t(\mathbf{x}) = \alpha^t (\mathbb{E}(h^t(\mathbf{x})) - \widehat{h}^t(\mathbf{x})). \tag{9}$$

It is important that $\widehat{h}^t(\mathbf{X})$ is the oracle estimator of $Y - \widehat{F}^{t-1}(\mathbf{X})$, and h^t is not defined as an empirical solution of this regression problem. Instead, h^t is the minimizer of $L^t(h) := \sum_j (h(\mathbf{X}_j) - (Y_j - M_{j-1}^{t-1}(\mathbf{X}_j))^2$. The key observation is that the model M_{j-1}^{t-1} and variable $\mathbf{X}_j$ are independent, since M_{j-1}^{t-1} is defined by $\{(\mathbf{X}_k, Y_k) \mid k = 1, \ldots, j-1\}$. Therefore, L_t has the same distribution as its altered version L_t' defined by $L_t'(h) := \sum_{j=1}^n (h(\mathbf{X}_{n+j}) - (Y_{n+j} - M_{j-1}^{t-1}(\mathbf{X}_{n+j})))$, and thus the minimizer h^t of L_t and the minimizer h'^t of L_t' have the same expectations:

$$\mathbb{E}(h^t) = \mathbb{E}(h'^t). \tag{10}$$

Consider an altered functional $L_t''(h) := \sum_{j=1}^n (h(\mathbf{X}_{n+j}) - (Y_j - \frac{1}{n} \sum_{s=1}^n M_{s-1}^{t-1}(\mathbf{X}_{n+j}))$.

On one hand, L_t'' can be viewed as the average $\frac{1}{n!} \sum_{\sigma \in S_n} L_{t,\sigma}'$ over symmetric group S_n, where $L_{t,\sigma}'(h) := \sum_{j=1}^n (h(\mathbf{X}_{n+\sigma(j)} - (Y_{n+\sigma(j)} - M_{j-1}^{I-1}(\mathbf{X}_{n+\sigma(j)}))))$. As long as H^t is linear, the minimizer h''^t of L_t'' can be derived as the convex combination $h''^t = \frac{1}{n!} \sum_{\sigma \in S_n} h_\sigma'^t$ of corresponding minimizers $h_\sigma'^t$ of $L_{t,\sigma}'$. Due to symmetry of $\{(\mathbf{X}_{n+j}, Y_j)\}$ with respect to j, we have $\mathbb{E}(h_\sigma'^t) = \mathbb{E}(h'^t)$ for any $\sigma \in S_n$. Therefore,

$$\mathbb{E}(h''^t(\mathbf{x})) = \frac{1}{n!} \sum_{\sigma \in S_n} \mathbb{E}(h_\sigma'^t(\mathbf{x})) = \mathbb{E}(h'^t). \tag{11}$$

On the other hand, h''^t is an empirical estimation of $Y^{t-1} := Y - \frac{1}{n} \sum_{s=1}^n M_{s-1}^{t-1}(\mathbf{X})$. By induction, $\mathbb{E}(M_{s-1}^{t-1}(\mathbf{x})) = \widehat{M}^{t-1}(\mathbf{x})$, and therefore $\mathbb{E}(Y^{t-1} \mid \mathbf{X} = \mathbf{x}) = \mathbb{E}(Y - \widehat{M}^{t-1}(\mathbf{X}) \mid \mathbf{X} = \mathbf{x})$, what implies that the oracle regression $\widehat{h}''^t$ of Y^{t-1} and the oracle regression $\widehat{h}^t$ for $Y - \widehat{M}^{t-1}(\mathbf{X})$ coincide: $\widehat{h}''^t = \widehat{h}^t$. Thus

$$\mathbb{E}(h''^t(\mathbf{x})) = \widehat{h}''^t(\mathbf{x}) = \widehat{h}^t(\mathbf{x}). \tag{12}$$

Now Theorem 1 follows from equations 9, 10, 11, and 12.

Description of datasets

Table 3: Description of datasets

Dataset	n	test set	m	features	task
Allstate	8.5M	4.7M (for 2007)	33	categorical, numerical	classification
Springleaf	116K	29K (random 20%)	1934	categorical, numerical	classification
Liberty	41K	10K (random 20%)	32	categorical, numerical	regression
Adult	39K	10K (random 20%)	14	categorical, numerical	classification
Abalone	3.3K	0.8K (random 20%)	8	numerical	regression