Tackling Overfitting in Boosting for Noisy Healthcare Data

Yubin Park, Joyce C. Ho, *Member, IEEE,*

Abstract—Analyzing healthcare data poses several challenges including the limited number of samples, missing measurements, noisy labels, and heterogeneous data types. Tree-based boosting is well-suited for modeling such data as it is insensitive to data types and missingness. Moreover, Stochastic Gradient TreeBoost is often found in many winning solutions in public data science challenges. Unfortunately, the best performance requires extensive hyperparameter tuning and can be prone to overfitting. We propose PaloBoost, a Stochastic Gradient TreeBoost model that uses novel regularization techniques to guard against overfitting and is robust to hyperparameter settings. PaloBoost uses the out-of-bag samples to perform gradient-aware pruning and estimate adaptive learning rates. Unlike other Stochastic Gradient TreeBoost models that use the out-of-bag samples to estimate test errors, PaloBoost treats the samples as a second batch of training samples to prune the trees and adjust the learning rates. As a result, PaloBoost can dynamically adjust tree depths and learning rates to achieve faster learning at the start and slower learning as the algorithm converges. Experimental results on four datasets demonstrate that PaloBoost is robust to overfitting and is less sensitive to the hyperparameters.

Index Terms—Gradient Boosting, Overfitting, Healthcare Data, Noisy Data

1 Introduction

Healthcare data is peculiar and unlike data used in many other applications of machine learning and data mining. The number of samples (patients) are often small compared to the number of features [1]. The data perpetually has missing measurements (e.g., blood tests are often not collected at every visit). The labels can be noisy due to the subjective nature of human judgment or even an incomplete representation of the patient [2], [3]. Finally, data is collected from multiple sources and consists of various data types [4]. Thus, predictive models face formidable challenges when analyzing healthcare data – complex models often suffer from overfitting while simple models do not provide much predictive power.

Tree-based boosting, or TreeBoost, is particularly well-suited for healthcare data. TreeBoost is a stage-wise additive model where base trees are fitted to the subsampled gradients (or errors) at each stage [5], [6]. Trees are appealing due to their insensitivity to data types and distributions [7]. Trees can also readily handle missing data without requiring additional preprocessing such as mean imputation [8]. Moreover, tree-based models are more readily interpretable and have been widely adopted in healthcare domains [9], [10].

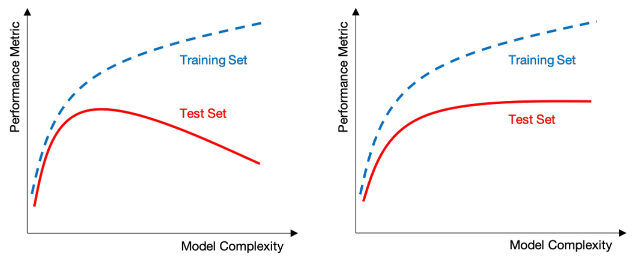
Stochastic Gradient TreeBoost (SGTB) is one of the most widely used off-the-shelf boosting algorithms [11], [12], [13]. The randomness introduced by subsampling speeds up the computation time and mitigates overfitting. SGTB achieves robust performance over various classification, regression, and even ranking tasks [14], [15]. Many empirical results have demonstrated SGTB's ability to model complex and large data relatively fast and accurately []. Furthermore, the effectiveness and pervasiveness of SGBT can be found in many winning solutions in public data science challenges [13].

While SGTB can generally provide reasonable performance with the default hyperparameter settings, to achieve its best performance usually requires extensive hyperparameter tuning. In general, there are four hyperparameters in SGTB to tune: the maximum depth of the tree, the number of trees, the learning rate, and the subsampling rate. To pick the best-performing hyperparameters, we need to further split the training data into two sets: one for fitting SGTBs for a range of hyperparameters, and the other for comparing the performance of SGTBs with different hyperparameters [16], [17], [18], [19], [20]. While the maximum depth, subsampling rate, and learning rate have general guidelines, each hyperparameter is not independent of one another. Having deeper trees and more trees can reduce training errors more rapidly, although they can increase the chance of overfitting at the same time [5], [21], [22]. Lower learning rates, on the other hand, can mitigate overfitting to the training data at each stage, but it needs more trees to reach a similar level of performance [21]. Thus the delicate balancing act between overfitting and best performance is more of an art than a science in practice.

Navigating the fine line between overfitting and best performance can be even more challenging when the best performance depends on a narrow range of hyperparameter configurations. The performance can substantially degrade if the hyperparameters are slightly off from the "optimal" region, as shown in Figure 1(a). Moreover, even if the optimal hyperparameters could be reliably estimated, there is no guarantee that the hold-out validation data is drawn from the exact same distribution as the test data – thereby increasing the chance of performance degradation [23], [24].

• *Y. Park is with Bonsai Research, LLC.*
E-mail: yubin@bonsairesearch.com
• *J. C. Ho is with Emory University.*
E-amil: joyce.c.ho@emory.edu

Manuscript received April 19, 2005; revised August 26, 2015.



(a) Hyperparameter Sensitive (b) Hyperparameter Robust

Fig. 1. Overfitting Behaviors of Hyperparameter-Sensitive (left) vs Hyperparameter-Robust Models (right). The performance of hyperparameter-robust models are less sensitive to the "optimal" hyperparameters and a minor distributional shift in test data.

Unfortunately, the described scenario is commonplace in healthcare applications. Healthcare data tend to have noisy labels, inaccurate data, small sample sizes, and a large number of features. Furthermore, predictive tasks in healthcare often involve some level of distributional shifts in test data[1]. Therefore, the development of a "hyperparameter robust" boosting algorithm that can achieve near-best performance under a broad range of hyperparameter configurations is necessary to mitigate these issues. A hyperparameter robust model, shown in Figure 1(b), can exhibit stable performance even if the model is very complex.

We present PaloBoost[2], a boosting algorithm that makes it easier to tune the hyperparameters and potentially achieve better performance. PaloBoost extends SGTB using two *out-of-bag sample* regularization techniques: 1) Gradient-aware Pruning and 2) Adaptive Learning Rate. Out-of-bag (OOB) samples, the samples not included from the subsampling process, are commonly available in many subsampling-based algorithms [25], [26]. In boosting algorithms, OOB errors, or the errors estimated from OOB samples, are used as computationally cheaper alternatives for cross-validation errors and to determine the number of trees, also known as early stopping [21]. However, OOB samples are under-utilized in SGTB. They merely play an observer role in the overall training process, ignored and unused.

PaloBoost, on the other hand, treats OOB samples as a second batch of training samples for adjusting the learning rate and max tree depth. At each stage of PaloBoost, the OOB errors are used to determine the generalization properties of the tree. If the OOB error does not decrease, the tree is too specific and likely overfits the in-bag samples. To mitigate the overfitting effect, the tree leaves are pruned to reduce the tree complexity, and optimal learning rates are estimated to control and decrease the OOB errors. Thus, at each stage, PaloBoost uses in-bag samples for learning the tree structure, and OOB samples to prune and adjust the learning rates. As a result, PaloBoost's hyperparameter tuning occurs at each stage with different OOB samples.

We compared the performance of PaloBoost with various open-source SGTB implementations including Scikit-Learn [27] and XGBoost [13]. Our benchmark datasets include two simulated datasets [8], [28] and 3 real healthcare datasets. The empirical results demonstrate stable, predictive performance in the presence of noisy features. PaloBoost is hyperparameter robust – considerably less sensitive to the hyperparameters. In addition, PaloBoost hardly suffers significant performance degradations with more trees. The results also illustrate the adaptive learning rates and tree depths that guard against overfitting. Thus, PaloBoost offers a robust, SGTB model that can save computational resources and remove the art from hyperparameter tuning.

2 Background

Boosting algorithms build models in a stage-wise fashion, where the sequential learning of base learners provides a strong final model. Breiman demonstrated that boosting can be interpreted as a gradient descent algorithm at the function level [29]. In other words, boosting is an iterative algorithm that tries to find the optimal "function", where the function is additively updated by fitting to the gradients (or errors) at each stage. Later, Friedman formalized this view and introduced Gradient Boosting Machine (GBM) that generalizes to a broad range of loss functions [5]. Stochastic Gradient TreeBoost (SGTB) further builds on GBM to provide better predictive performance. In this section, we provide the formulation and basics of GBM and illustrate how SGTB is derived.

2.1 Gradient Boosting Machine

GBM [5] seeks to estimate a function, F^*, that minimizes the empirical risk associated with a loss function $L(\cdot,\cdot)$ over N pairs of target, y, and input features, $\mathbf{x}$:

$$F^* = \arg\min_F \sum_{i=1}^{N} L(y_i, F(\mathbf{x}_i)). \tag{1}$$

Examples of frequently used loss functions are squared error, $L(y,F) = (y-F)^2$ and negative binomial log-likelihood, $L(y,F) = -yF + \log(1+e^F)$, where $y \in \{0,1\}$. GBM solves Equation (1) by applying the gradient descent algorithm directly to the function F. Each iterative update of the function F has the form:

$$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \beta_m \frac{\partial}{\partial F(\mathbf{x})} L(y, F(\mathbf{x}))\Big|_{F=F_{m-1}}, \tag{2}$$

where β_m is the step size. With finite samples, we can only approximate the gradient term with an approximation function, $h(\mathbf{x})$:

$$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \beta_m h(\mathbf{x}; \mathbf{a}_m), \tag{3}$$

where $\mathbf{a}_m$ represents the parameters for the approximation function at iteration m. In GBM, $h(\mathbf{x};\mathbf{a})$ can be any parametric function such as neural net [30] and regression tree [5].

An alternative perspective of GBM is to view it as an ensemble where the approximation function, $h(\mathbf{x})$, is the

1. Some examples of such distributional shifts in test data are 1) applying a pre-trained readmission prediction model at a newly built hospital, 2) applying a clinical risk engine for a state-level Medicaid plan that may change its eligibility criteria every so often, and 3) applying a utilization prediction engine for commercial health plan that may change coverage and prior-authorization policies for specific treatments.

2. "Palo" in PaloBoost stands for **P**runing and **A**daptive **L**earning with **O**ut-of-Bag Samples.

base learner and β_m represents the corresponding weight. The GBM model produces a final output F of the form:

$$F(\mathbf{x}) = F_0 + \sum_{m=1}^{M} \beta_m h_m(\mathbf{x}). \tag{4}$$

where M is the maximum number of iterations (or trees). XGBoost assumes this ensemble form F to derive the optimal base learners by applying the Taylor approximation and a greedy optimization technique [13]. However, PaloBoost is better understood in the context of the original GBM work [5].

2.2 Stochastic Gradient TreeBoost

Stochastic Gradient TreeBoost (SGTB) introduces three important modifications to GBM: (1) tree structure-aware (or node-wise) step sizes, (2) subsampling at each stage, and (3) shrinkage technique (i.e. learning rate) [5]. Friedman observed that a tree partitions the input into J disjoint regions ($\{R_j\}_1^J$), where each region predicts a constant value (b_j). Thus, the approximation function can be expressed as:

$$h(\mathbf{x}; \mathbf{a}) = h(\mathbf{x}; \{b_j, R_j\}_1^J) = \sum_{j=1}^{J} b_j \mathbb{1}(\mathbf{x} \in R_j). \tag{5}$$

This representation allows each region to choose its own optimal step size, β_{jm}, instead of a single step size per stage β_m to obtain a better predictive model. Therefore, Equation (3) can be rewritten as:

$$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \sum_{j=1}^{J} \gamma_{jm} \mathbb{1}(\mathbf{x} \in R_{jm}) \tag{6}$$

where $\gamma_{jm} = \beta_{jm} b_j$. As a result, estimating γ_{jm} is equivalent to estimating the intercept that minimizes the loss function within the disjoint region R_{jm}:

$$\gamma_{jm} = \arg\min_{\gamma} \sum_{\mathbf{x}_i \in R_{jm}} L(y_i, F_{m-1}(\mathbf{x}_i) + \gamma). \tag{7}$$

The second important modification in SGTB is subsampling, which is motivated by bagging [25] and AdaBoost [31]. Friedman found that random subsampling of the training samples greatly improved the performance and generalization abilities of GBM, while reducing the computing time. Although the performance improvement varied across problems, he observed that subsampling has a greater impact on small datasets with high capacity base learners. Consequently, he postulated that the performance improvement may be due to the variance reduction as in bagging. Typical subsampling rates (q) lie between 0.5 and 0.7.

Lastly, a shrinkage technique was introduced to scale the contribution of each tree by a factor ν between 0 and 1. This parameter can also be viewed as the learning rate ν in the context of stochastic gradient descent. Lower ν values (≤ 0.1) slow down the learning speed of SGTB and necessitate more iterations, but may achieve better generalization [6]. Algorithm 1 illustrates the details of SGTB with a learning rate ν. Thus, SGTB has four different parameters that require tuning: (1) tree size (J); (2) number of trees (M); (3) the subsampling rate (q); and (4) the learning rate (ν).

Algorithm 1 Stochastic Treeboost with Learning Rate (ν)

```
$F_0 = \arg\min_\beta \sum_{i=1}^N L(y_i, \beta)$
for $m = 1$ to $M$ do
    $\{y_i, \mathbf{x}_i\}_1^{N'} = \text{Subsample}(\{y_i, \mathbf{x}_i\}_1^N, \text{rate} = q)$
    for $i = 1$ to $N'$ do
        $z_i = -\frac{\partial}{\partial F(\mathbf{x}_i)} L(y_i, F(\mathbf{x}_i))\Big|_{F=F_{m-1}}$
    end for
    $\{R_{jm}\}_1^J = \text{RegressionTree}(\{z_i, \mathbf{x}_i\}_i^{N'})$
    for $j = 1$ to $J$ do
        $\gamma_{jm} = \arg\min_\gamma \sum_{\mathbf{x}_i \in R_{jm}} L(y_i, F_{m-1}(\mathbf{x}_i) + \gamma)$
    end for
    $F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \nu \sum_{j=1}^J \gamma_{jm} \mathbb{1}(\mathbf{x} \in R_{jm})$
end for
```

SGTB is widely available in many easy-to-use open-source packages. Some implementations include Generalized Boosted Models available in `R` [21], TreeNet by Salford Systems [32], Gradient Boosting in Scikit-Learn [27], and XGBoost implemented in `C++` [13]. Although all packages generally follow Algorithm 1, the implementation details are slightly different. As an example, the splitting criteria of the base learner varies across packages – Generalized Boosted Models uses the minimum variance criterion, Scikit-Learn uses the Friedman splitting criterion [5] by default, and XGBoost uses a custom regularized splitting criterion that is obtained by greedily minimizing the Taylor approximation of the loss function [13]. The treatment of missing values can also differ (e.g., XGBoost can handle missing values by default, while Scikit-Learn does not support it). Thus, the performance can vary across packages even when applied to the same dataset. No conclusive evidence can be drawn that one implementation is the best as it depends on the dataset.

3 PaloBoost

The performance of SGTB is dependent on its hyperparameters: tree depth, number of trees, subsampling rate, and learning rate. Unfortunately, each hyperparameter is not independent of the others and makes the tuning process quite difficult. As an example, the ν-M trade-off [5], [6], captures the relationship between the learning rate and the number of trees. Lower learning rates (ν) may result in better performance, but more trees (M) are usually required to achieve a similar level of performance. Thus, an exhaustive grid search is often needed to find the optimal hyperparameters.

Although the hyperparameter tuning process is relatively straightforward, it can consume a substantial amount of computational resources. We illustrate the tuning process for tree depth and learning rate for a *single-stage* SGTB model (i.e., $M = 1$). The dataset is partitioned into three sets: training, validation, and test. The test set is strictly set aside and used to estimate the generalization error. First, the intercept term, F_0, is fit using the training set. Next, for a given learning rate and tree depth, we randomly subsample data points from the training set and fit a tree on those sampled gradients. The performance is then measured on the validation set. This process is repeated multiple times using varying learning rates and tree depths. The hyperparameters are then chosen based on the best validation performance. The sample principle is applied for SGTBs with multiple stages, where the number of trees (M) is also varied. As a result, the whole cycle, while parallelizable, is computationally expensive and requires considerable time.

Unfortunately, even the best-tuned hyperparameters may not produce optimal results in real testing environments. Since the training and validation samples may not reflect the "true" distribution, there is a real danger of overfitting. Thus, the hyperparameters chosen via the expensive tuning process may not yield a robust, and generalizable model. We have developed PaloBoost, a variant of SGTB that is less sensitive to hyperparameters and provides robust predictive performance. PaloBoost eliminates the need to finely tune the learning rate and tree depth, and instead estimates them during the training process.

PaloBoost mitigates overfitting via adaptive learning rates[3] and tree depths. The main idea centers around the *under-utilized* out-of-bag (OOB) samples to tune these hyperparameters on the fly. We observed that *both validation and OOB samples* are not seen during each tree building stage. Thus, the OOB samples can be used to estimate the learning rate and the tree depth at each stage. The same principle can be applied at the next tree building stage even though the OOB samples are different, as the learning rate and tree depth are stage-specific parameters in PaloBoost. Additionally, the stage-specific adaptive learning rates serve as a guard against overfitting and can be used to determine the optimal number of trees. Thus, PaloBoost does not need to maintain a separate validation set, thereby increasing the number of training samples at each stage that can further combat tree overfitting.

3.1 Gradient-Aware Pruning

The maximum depth of the trees is a hyperparameter for many SGTB implementations. While the tree at each stage may not grow to the maximum depth for various reasons (e.g., insufficient samples in the node, information gain is not above a tolerance, etc.), the "intention" is to maintain the maximum height. Consequently, the majority of the trees in SGTB implementations will be grown to the maximum tree depth and can result in overfitting. Although XGBoost has a pre-pruning regularization hyperparameter [13], our empirical studies showed little impact unless the default hyperparameter was adjusted substantially. Instead, we introduce a gradient-aware pruning technique that utilizes the OOB samples to achieve more flexible tree depths. Even though maximum depth remains a hyperparameter in PaloBoost, our model is less sensitive to the value and does not need to be finely tuned.

Gradient-aware pruning has roots in the bottom-up "reduced-error pruning" found in decision tree literature. In reduced-error pruning, the errors on OOB samples are compared between children and parent nodes. If the child node does not decrease the error, the node is pruned, thereby reducing the complexity of the tree. However, boosting poses two challenges: each tree is dependent on the previous trees, and the node estimates are always multiplied by the learning rate.

Conceptually, gradient-aware pruning removes gradient estimates that do not generalize well to other samples. As the tree depth increases, the number of samples per node decreases. Smaller samples result in higher variances of the estimated gradient, γ, which is also likely to increase generalization errors[4]. Thus, to achieve more stable gradient estimates, the regions with high variance should be merged. In PaloBoost, after a tree is fitted to the subsampled gradients, the tree is applied to the OOB samples. For each disjoint region of the tree (R_j), the loss associated with introducing a new leaf estimate is the gradient multiplied by the learning rate:

$$\text{Loss}(R_j) = \sum_{i}^{L_j} \text{Loss}(y_i, \hat{y}_i + \nu\gamma_j) \tag{8}$$

Thus, if the leaf estimate does not reduce the loss on the OOB samples, the node should be pruned. The gradient-aware pruning process is summarized in Algorithm 2, where the distribution in the Algorithm refers to the distribution of the noise applied to the target i.e. "gaussian" and "bernoulli" correspond to regression and classification tasks, respectively.

Algorithm 2 Gradient-Aware Pruning (GAP)

```
$\{(R_j, R_k)\}$ = Find-Sibling-Pairs($\{R_j\}_{j=1}^{J}$)
for each sibling pair in $\{(R_j, R_k)\}$ do
    $\{y_i, \mathbf{x}_i\}_1^{L_j}$ = Out-of-Bag($\{y_i, \mathbf{x}_i\}_1^{N} | R_j$)
    $\{y_i, \mathbf{x}_i\}_1^{L_k}$ = Out-of-Bag($\{y_i, \mathbf{x}_i\}_1^{N} | R_k$)
    if $\sum_i^{L_j} \text{Loss}(y_i, \hat{y}_i) < \sum_i^{L_j} \text{Loss}(y_i, \hat{y}_i + \nu_{max}\gamma_j)$ then
        do_merge = True
    else if $\sum_i^{L_k} \text{Loss}(y_i, \hat{y}_i) < \sum_i^{L_k} \text{Loss}(y_i, \hat{y}_i + \nu_{max}\gamma_k)$ then
        do_merge = True
    else
        do_merge = False
    end if
    if do_merge then
        Merge($R_j$, $R_k$)
    end if
end for
```

3.2 Adaptive Learning Rate

The learning rate, ν, controls the contribution of each new stage. Empirically, the best strategy is to set ν to a very small value to achieve favorable test errors at the cost of larger values of M [8]. However, a single learning rate for every tree may not be optimal, as some trees do not generalize well. Instead, each stage should have a different learning rate. Unfortunately, calculating optimal stage-specific learning rates through standard optimization techniques (i.e., line search) introduces substantial computational overhead.

3. The adaptive learning rates in PaloBoost are not based on the gradients, which are commonly referred to as adaptive learning rates in deep learning models.

4. Decision trees are often viewed as a high variance model. By merging two siblings, there will be more samples and thereby reducing the variance of the node estimate.

Rather, PaloBoost takes advantage of the tree structure to calculate optimal learning rates for each region efficiently.

The key observation is that we can decouple the estimation of the learning rate if we introduce a region-specific learning rate at each stage. Thus, each region R_j can calculate the multiplicative factor that optimizes the OOB loss:

$$\nu_j^* = \arg\min_{\nu} \sum_{i}^{L_j} Loss(y_i, \hat{y}_i + \nu\gamma_j), \tag{9}$$

where L_j represents the number of OOB samples in the leaf of interest. The benefit of this approach is that closed-form solutions exist for many loss functions. For example, the learning rate for the negative binomial log-likelihood loss function is:

$$\nu_j^* = \arg\min_{\nu} \sum_{i}^{L_j} \log(1 + \exp(\hat{y}_i + \nu\gamma_j)) - y_i(\hat{y}_i + \nu\gamma_j) \tag{10}$$

$$= \log\left(\frac{\sum_i^{L_j} y_i}{\sum_i^{L_j} (1 - y_i)\exp(\hat{y}_i)}\right) / \gamma_j \tag{11}$$

Similarly, for the squared error loss function, the closed form solution is:

$$\nu_j^* = \arg\min_{\nu} \sum_{i}^{L_j} (y_i - (\hat{y}_i + \nu\gamma_j))^2 \tag{12}$$

$$= \frac{\sum_{i=1}^{L_j}(y_i - \hat{y}_i)}{\gamma_j L_j} \tag{13}$$

PaloBoost also introduces a clipping function on the estimated learning rate, to reduce the effect of small OOB sample sizes. Without clipping, the estimated learning rates can fluctuate in a wide range, sometimes exceeding the value 1. Thus, to enforce stability and maintain similarity with existing SGTB implementations, estimated learning rates are capped with the maximum learning rate hyperparameter ν_{max}. As a result, the effective region-specific learning rate ranges between zero and the maximum specified learning rate. Algorithm 3 illustrates our adaptive learning rate strategy.

Algorithm 3 Adaptive Learning Rate (ALR) with Out-of-Bag Loss Reduction

```
$\{(R_j, R_k)\}$ = Find-Sibling-Pairs($\{R_j\}_{j=1}^J$)
for $R_j$ in $\{R_j\}_{j=1}^J$ do
    $\{y_i, \mathbf{x}_i\}_1^{L_j}$ = Out-of-Bag($\{y_i, \mathbf{x}_i\}_1^N | R_j$)
    if distribution=="gaussian" then
        $\nu_j = \text{clip}\left(\frac{\sum_{i=1}^{L_j}(y_i - \hat{y}_i)}{\gamma_j L_j}, 0, \nu_{max}\right)$
    else if distribution=="bernoulli" then
        $\nu_j = \text{clip}\left(\log\left(\frac{\sum_i^{L_j} y_i}{\sum_i^{L_j}(1-y_i)\exp(\hat{y}_i)}\right)/\gamma_j, 0, \nu_{max}\right)$
    end if
end for
```

Gradient-aware pruning (GAP) serves as a preprocessing step for the adaptive learning rate mechanism. Removing the regions with higher variances of node estimates (γ) provides two main benefits: (1) there are less adaptive rates to estimate, and (2) robust node estimates will yield better learning rates. Without GAP, PaloBoost still can dynamically adjust learning rates with the ALR algorithm, but such adjusted learning rates tend to exhibit more variance. In short, we need both GAP and ALR for PaloBoost to produce hyperparameter-robust models. Algorithm 4 provides the details of PaloBoost. As the learning rate and tree depth values are re-estimated during the training process, there is less sensitivity to the maximum learning rate hyperparameter ν_{max} and tree depth.

Algorithm 4 PaloBoost

```
$F_0 = \arg\min_\beta \sum_{i=1}^N L(y_i, \beta)$
for $m = 1$ to $M$ do
    $\{y_i, \mathbf{x}_i\}_1^{N'}$ = Subsample($\{y_i, \mathbf{x}_i\}_1^N$, rate $= q$)
    for $i = 1$ to $N'$ do
        $z_i = -\frac{\partial}{\partial F(\mathbf{x}_i)} L(y_i, F(\mathbf{x}_i))\Big|_{F=F_{m-1}}$
    end for
    $\{R_{jm}\}_1^J$ = RegressionTree($\{z_i, \mathbf{x}_i\}_i^{N'}$)
    for $j = 1$ to $J$ do
        $\gamma_{jm} = \arg\min_\gamma \sum_{\mathbf{x}_i \in R_{jm}} L(y_i, F_{m-1}(\mathbf{x}_i) + \gamma)$
    end for
    $\{R_{jm}, \gamma_{jm}\}_1^{J'}$ = GAP($\{R_{jm}, \gamma_{jm}\}_1^J, \nu_{max}$)
    for $j = 1$ to $J'$ do
        $\nu_{jm}$ = ALR($\gamma_{jm}, \nu_{max}$)
    end for
    $F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \sum_{j=1}^J \nu_{jm}\gamma_{jm}\mathbb{1}(\mathbf{x} \in R_{jm})$
end for
```

These two modifications to SGTB (GAP and ALR), can make SGTB more robust to hyperparameters at a very small price in its training time. To analyze its time complexity, consider a dataset with n samples and m columns. For fitting a decision tree with a maximum depth of d, it takes $O(nmd)$ in time. During the construction of each base tree in PaloBoost, GAP and ALR re-visits every node of the base tree at most once respectively. Thus, in the worst case, GAP and ALR requires an additional scan of the data samples, which is equivalent to $O(2n)$. Therefore, each tree in PaloBoost has the time complexity of $O(nmd + 2n)$. This added cost becomes marginal in the presence of a large number of features (i.e. $m \gg 2$), which is quite common in many real-world data problems.

3.3 Modified Feature Importance

Feature importance in SGTB is calculated as the summation of squared error improvements by the feature [33]. Although the formula is adopted from RandomForest [26], the measure is purely based on heuristic arguments [5]. While the feature importance has proven useful in many applications, we notice that it ignores two important aspects of SGTB: the leaf estimates and the coverage of the region (i.e., number of samples in the node). As seen in the SGTB algorithm (Algorithm 1), leaf estimates are recalibrated to minimize the loss function. Moreover, the improvement from a feature at a particular branch may yield leaves that have extremely small coverage. Thus, the feature importance may not be an accurate indication of coverage and impact.

These issues, along with the new adaptive learning rate, led us to create a new feature importance formula. Instead of a branch-centric perspective, PaloBoost introduces a leaf-centric formula that accounts for leaf estimate, region coverage, and region-specific learning rate. The importance of a feature, f_k is calculated as:

$$f_k = \sum_{m=1}^{M} \frac{\sum_{j=1}^{J_m} \nu_{jm}|R_{jm}||\gamma_{jm}|\mathbb{1}(f_k \in \text{Rules}_{jm})}{J_m \sum_{j=1}^{J} R_{jm}} \tag{14}$$

where $|R_{jm}|$ represent the coverage of the region R_{jm}, and Rules_{jm} denotes the logical rules that define the region R_{jm}. For example, if Rules_{jm} is of the form ($x_0 > 0.5$ AND $x_1 < 1.0$ AND $x_3 > -1$), then $\mathbb{1}(f_1 \in \text{Rules}_{jm})$ will be one as x_1 is in the rules. With our feature importance formula, if a feature is used to define a region, we will multiply the size of the region ($|R_{jm}|$) with the effective absolute node estimate, $\nu_{jm}|\gamma_{jm}|$. Therefore, if any of the three quantities (coverage of the region, the leaf estimate, or the adaptive learning rate) is small, the feature importance contribution is minimal.

We found that this modified feature importance can differentiate meaningful features from noisy features better than the original feature importance using a simulated data set. Some experimental results are available in [34]. However, in this paper, we did not include the results to focus on our results regarding overfitting and predictive performance.

4 Experimental Results

PaloBoost is implemented in Python using the Bonsai Decision Tree framework [35], which allows easy manipulation of decision trees. The Bonsai framework is mostly written in Python with its core computation modules written in Cython, to provide C-like performance. Our implementation of PaloBoost is made publicly available as an extension to Bonsai[5].

PaloBoost is evaluated on a variety of datasets (simulated and real-world healthcare data) and compared against state-of-the-art SGTB implementations. In this section, we will analyze the following aspects of PaloBoost:

- How different are the learning rates and tree depths at each individual stage?
- How sensitive is PaloBoost to hyperparameters?
- How does the predictive performance compare with other SGTB implementations?

4.1 Datasets

Five different datasets are used to evaluate the SGTB variants, including PaloBoost. We use two simulated datasets and three publicly available datasets, two from Physionet 2012 Challenge [36], and one from MIMIC-III [37]. Our benchmark study consists of two regression tasks and three classification tasks. Table 1 lists the datasets and their associated characteristics. A brief overview of each dataset will be provided in the context of their respective tasks.

All datasets are preprocessed in a uniform way: 1) no imputation for missing values, 2) minimal outlier detection, 3) adding polynomial (or interaction) features for better predictive performance. While XGBoost and our SGTB implementations (including PaloBoost) can naturally handle missing values, we omit Scikit-Learn for datasets with missing values as it does not support missing values. We made all the scripts used in the experiments publicly available. For more details, please see https://github.com/yubin-park/bonsai-dt/tree/master/research.

5. The implementation of PaloBoost can be found in https://yubin-park.github.io/bonsai-dt/ as one of the Bonsai templates.

4.2 Baselines

We compare PaloBoost with three other baseline models as follows:

- Scikit-Learn [27], the de-facto machine learning library in Python that has been widely adopted.
- XGBoost [13], the battle-tested library that has won many data science challenges.
- Bonsai-SGTB, an SGTB implementation using the Bonsai framework.

While there are many newly developed SGTB implementations (e.g., LightGBM, CatBoost), they are not included in our study for several reasons. First, we observed that the overfitting behavior of these packages is similar to Scikit-Learn and XGBoost as they focus primarily on implementation details and feature engineering. CatBoost focuses on more efficiently encoding categorical variables using Target-based Statistics and permutation techniques [40], [41]. LightGBM primarily focuses on training speed using less memory in distributed settings [42]. Secondly, our objective is to characterize the behaviors of the two regularization techniques in PaloBoost, rather than to claim that PaloBoost outperforms other SGTB implementations. We note that our regularization techniques, Gradient-aware Pruning and Adaptive Learning Rate, can be added to any existing SGTB implementations. Therefore, the three baseline models serve as suitable representations for other implementations.

4.3 Setup & Evaluation Metrics

All datasets are split with an 80/20 train-test ratio (i.e., 80% for training and 20% for test). For each SGTB implementation, we tested three different learning rates that spanned the spectrum for ν (1.0, 0.5, and 0.1), and three different tree depths of 3, 5 and 7. Unless otherwise specified, default hyperparameter settings were used for the benchmark models. We constructed 10 sets of random training-test splits, and each model was trained on 80% of the data and then applied to the test set to measure predictive performance. Thus, for each dataset, we have 10 runs for every 9 different hyperparameter settings (3 learning rates $\times$ 3 tree depths). We evaluated the models using the coefficient of determination (R^2) and the Area Under the Receiver Operating Characteristic Curve (AUROC) for the regression and classification tasks, respectively.

4.4 Simulated Datasets

4.4.1 Friedman Regression Dataset

Description. The simulated dataset contains 10,000 samples and is adopted from Friedman [28], [43], to introduce noisy

TABLE 1
Benchmark Datasets. For regression tasks, we list the standard deviation of the target variable, whereas the class ratios for classification tasks. The number of features shows the number after preprocessing e.g. removing small variance features, adding polynomial features.

Source	Dataset	Task	Missing Data	Target Stats	# Samples	# Features
Simulation	Friedman [28]	Regression	None	$\sigma(y) = 6.953$	10,000	55
Physionet 2012 [36]	Length of Stay in ICU [38]	Regression	Yes	$\sigma(y) = 12.20$	3,940	1,378
Simulation	Hastie [8]	Classification	None	$\mu(y) = 0.50$	10,000	66
Physionet 2012 [36]	Mortality in ICU [38]	Classification	Yes	$\mu(y) = 0.14$	3,940	1,378
MIMIC-III [37]	Cardiac Arrest 6Hr [39]	Classification	Yes	$\mu(y) = 0.27$	1,081	78

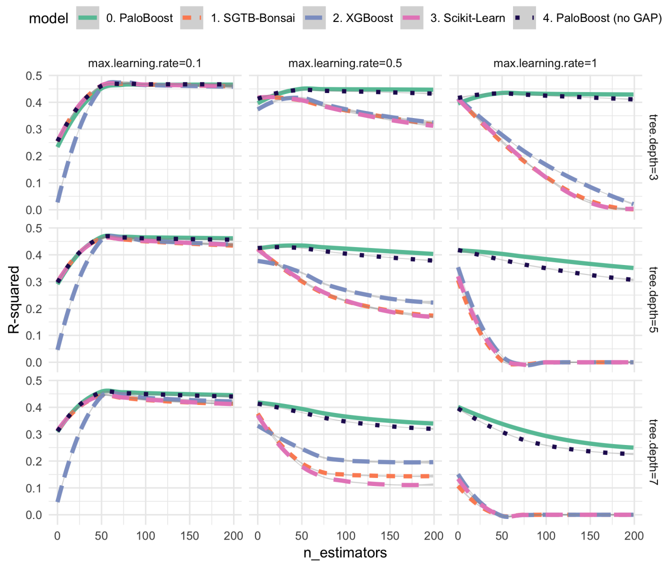


Fig. 2. [Friedman] Predictive performance over boosting iterations. Except for PaloBoost, the best performances are localized in a narrow region of hyperparameters. On the other hand, PaloBoost can maintain its near-best performances over many different hyperparameter settings.

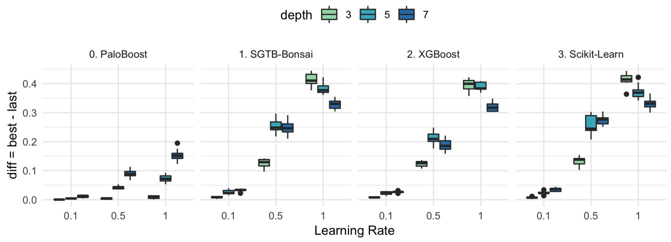


Fig. 3. [Friedman] Performance differences between the best and last boosting iterations. PaloBoost shows minimal performance differences over different hyperparameter configurations. This implies that PaloBoost can maintain its best performance even with many boosting iterations.

features (i.e., features with zero importance). The formula for the dataset is as follows:

$$y = 10\sin(\pi x_0 x_1) + 20(x_2 - 0.5)^2 + 10x_3 + 5x_4 + 5\epsilon \quad (15)$$

$$\epsilon \sim \text{Normal}(0, 1) \quad (16)$$

where x_is are uniformly distributed on the interval [0,1]. For each sample, Gaussian noise ϵ with a standard deviation of five, is added. Note that only 5 features among 10 features ($x_0 - x_4$) are actually used to generate the target (y) in Equation (15).

Predictive Performance. Figure 2 shows the coefficient of determination (R^2) over boosting iterations for the three different learning rates (in rows) and three tree depths (in columns). The curves are drawn by averaging the curves from the 10 random train-test runs. The horizontal axis on each cell represents boosting iterations, while the vertical axis indicates the coefficient of determination (R^2). The higher the R^2 values, the more accurate the predictions are. As can be seen, PaloBoost exhibits the best and most stable predictive performance for all hyperparameter configurations, more noticeably for the cases when high learning rates and deeper trees are used. While the other SGTB implementations show significant degradation in predictive performance as they iterate, PaloBoost displays a graceful drop of predictive performance even with high learning rates. Even for a small learning rate of 0.1, the overfitting behavior of SGTB becomes apparent after 50 iterations for tree depths of 5 and 7. Moreover, the predictive performance of PaloBoost varies considerably less across the three different learning rates, illustrating better robustness to the hyperparameter setting (ν_{max}). In the figure, we also included the performance of PaloBoost without GAP (dotted black line). We can observe that the effect of GAP is more noticeable in higher depth base trees, such as depth 5 and 7. GAP acts as a preprocessing step for ALR, and improve the overall performance across different hyperparameter settings.

Figure 3 shows the performance differences between the best and last boosting iterations. From left to right, each cell represents PaloBoost, Bonsai-SGTB, XGBoost, and Scikit-Learn models, respectively. In each cell, boxplots show the performance differences from different hyperparameter configurations. The smaller the performance differences, the less performance degradation from their best performances. As can be seen, PaloBoost shows minimal performance differences over different hyperparameter configurations. This suggests that PaloBoost can maintain its best performance even with many boosting iterations, while the other SGTB implementations overfit as they added more base trees.

Average Learning Rate and Pruning Rate. The gradual decline of predictive performance in PaloBoost can be better understood by analyzing the pruning rate and average learning rate at each stage. Figure 4 and 5 display the average learning rates and the pruning rates over the same iterations, respectively. On top of the original data in gray lines, we overlaid LOESS (locally estimated scatterplot smoothing) curves to visualize the overall trend using ggplot2 in R. From the figures, we can see a high variance in the learning rate for each stage and that it is often below the specified maximum rate. We can also observe that as the number of stages increases, the learning rates are adaptively adjusting to smaller values. Interestingly, some of the average learning rates at later iterations are near zero, meaning that the base trees hardly contributed to predictions. In addition, the pruning rate increases to yield lower variance trees. Thus,

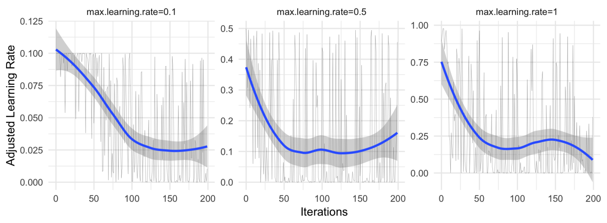


Fig. 4. [Friedman] The average learning rates over boosting iterations. Note that some of the average learning rates are near zero.

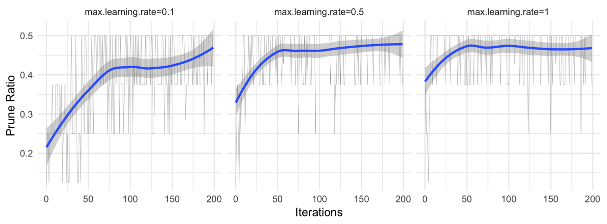


Fig. 5. [Friedman] The average pruning rates over boosting iterations. Base trees get pruned more as PaloBoost iteresates more.

these two mechanisms guard against overfitting even when adding more boosting iterations i.e. more complex models.

Feature Importance. Since the dataset is simulated, the true feature importance is known, with only 5 features ($x_0 - x_4$) used to generate the target. For the purpose of this experiment, we did not use any polynomial features as polynomial features mix up true and noisy features. Therefore, we have 10 variables instead of 55 variables for this specific experiment. We compared the feature importances from the four different trained models with 200 iterations ($M = 200$) and a learning rate of 0.5 ($\nu_{max} = 0.1$). PaloBoost and SGTB-Bonsai use the proposed feature importance formula (see Section 3.3), while XGBoost and Scikit-Learn use the standard feature importance formula (summation of the squared error improvements per feature). Figure 6 shows the feature importances from all four SGTB models[6]. PaloBoost clearly identifies the noisy features ($x_5 - x_9$), while XGBoost and Scikit-Learn estimate similar importances between the relevant and noisy features. We note that this result is partially due to the new importance formula, as we can see the feature importances of SGTB-Bonsai, which is similar to Scikit-Learn and XGBoost, are somewhat better than the other two SGTB algorithms. Although preliminary, this indicates PaloBoost can be also useful for the feature selection processes.

6. The results are similar for $M = 50$ but less dramatic.

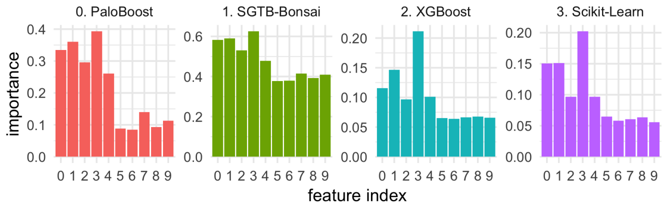


Fig. 6. The feature importances after 200 iterations. Only the first five features, x_0 to x_4, are used to generate the target.

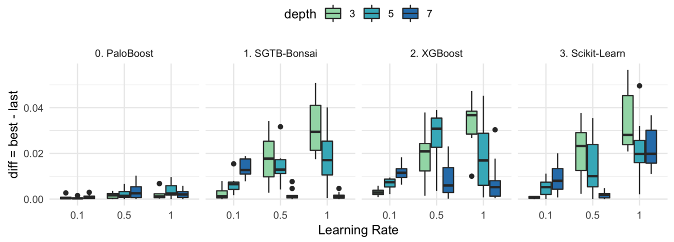


Fig. 7. [Hastie] Performance differences between the best and last boosting iterations. PaloBoost maintains its near-best performance even after reaching to its best performance.

4.4.2 Hastie Classification Dataset

This simulated dataset is derived from Example 10.2 in [8]:

$$Y = 1 \text{ if } \sum_{j=1}^{10} X_j^2 > \chi_{10}^2(0.5), \text{ otherwise -1} \tag{17}$$

As can be seen, the original model is fairly simple to fit, and it is difficult to observe overfitting behaviors. For the purpose of our experiment, we add an additional step to make the dataset more difficult to train and easier to overfit as follows:

$$Y' = 1 \text{ if } Z \times Y < \theta, \text{ otherwise } 0 \tag{18}$$

where Y' is the new target variable, $\theta = 0.2$, and $Z \sim \text{Normal}(0, 1)$ in this experiment. In other words, we added a multiplicative noise variable, Z, and flipped the class label with a threshold (θ).

Figure 7 shows the performance differences between the best and last boosting iterations. Similar to the results in the Friedman data experiment (Figure 3), PaloBoost shows minimal performance differences over various hyperparameter settings.

Figure 8 shows the predictive performance over boosting iterations. From the figure, we can easily observe that the other SGTB implementations are overfitting as they iterate more. It is noteworthy to mention the case where PaloBoost did not perform as good as the other algorithms i.e. tree depth=3 and learning rate=0.1. As can be seen, PaloBoost has not yet converged to the best performance, while other algorithms have seemingly approached the maximum performances. This is because the OOB regularization techniques can make PaloBoost converge slower than the others, since the “effective” learning rates and tree depth are often less than the specified values in the OOB regularization schemes. When a predictive task does not suffer from overfitting, we have observed that PaloBoost sometimes need more iterations to converge than the other SGTB implementations. The advantages of PaloBoost are better highlighted when we need to model a very noisy and overfit-prone dataset, hence healthcare datasets are the perfect applications of PaloBoost.

4.5 Real-World Healthcare Datasets

Length of Stay in ICU. Our first real-world healthcare dataset is from the Physionet 2012 challenge [36]. The dataset contains 4000 intensive care unit (ICU) patients and their physiological measurements during the first 48 hours after their ICU admissions. We chose this dataset because

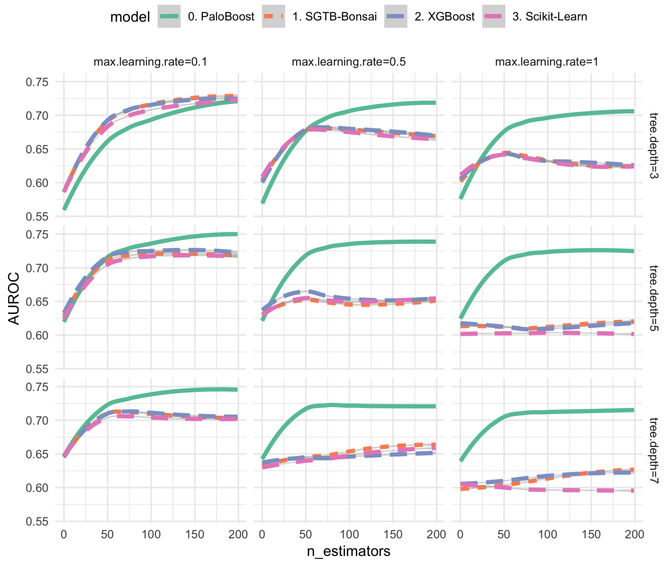


Fig. 8. [Hastie] Predictive performance over boosting iterations. PaloBoost's performance curves are fairly consistent regardless of hyperparameter settings.

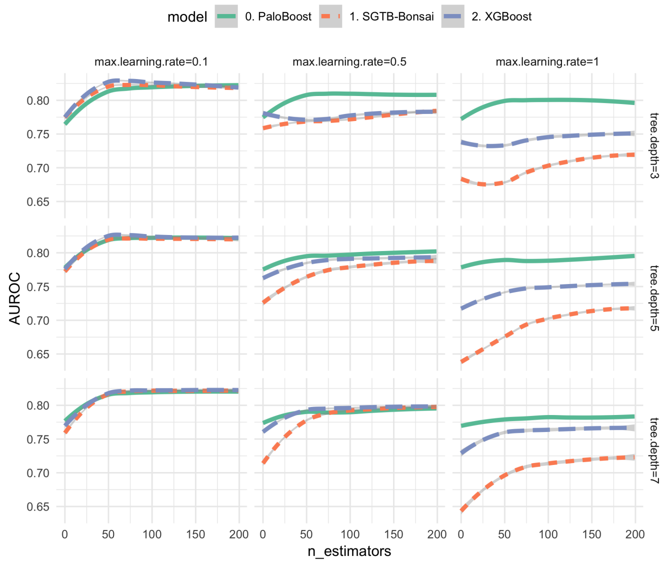


Fig. 10. [Mortality] Predictive performance over boosting iterations from the Mortality Prediction Task.

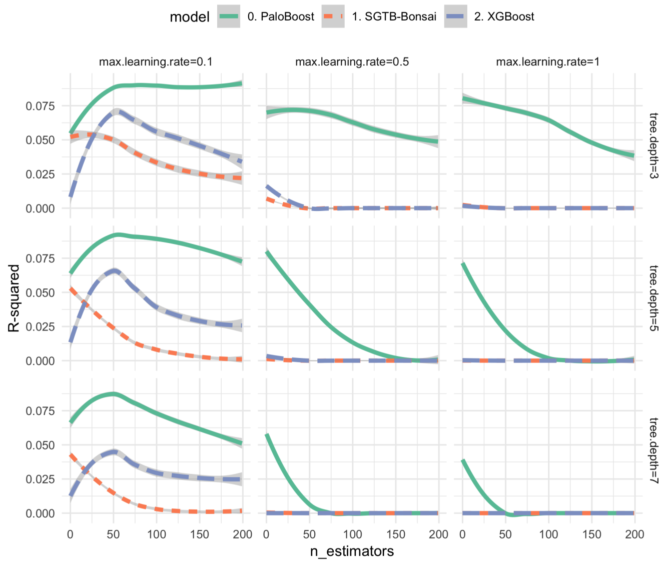


Fig. 9. [Length of Stay] Predictive performance over boosting iterations from the Length of Stay Prediction Task.

it is publicly available and readily accessible for replication. To minimize the presence of missing values, we pick the seven most commonly available measurements: noninvasive diastolic arterial blood pressure, temperature, heart rate, Glasgow Coma Score, serum glucose, white blood cell count and urine output. The feature matrix was constructed using summary statistics (average, minimum, maximum, first, and last measurements) for each of the measurements to yield a total of 35 features. 15.3% of the patients had at least one missing value and 60 patients who had no observations for any of the 7 measurements are not included in our study. See [38] for more details on data preparation.

The task for this experiment is to predict the total length of stay. This is extremely important for the hospital given the high equipment costs and the large percentage of medical resources used by ICU patients. Accurate predictions on when a bed in the ICU will be available can lead to considerable resource and management optimization. Furthermore, such predictions can also help physicians intervene in a more timely manner and design better care plans. This can lead to a rise in patient satisfaction, ultimately resulting in a higher reputation and credibility for the hospital as well.

Figure 9 shows the results of predicting the length of stay. The rows show different learning rates (0.1, 0.5, and 1.0 from left to right), and the columns represent different tree depths (3, 4, and 5). The horizontal axis on each cell is boosting iterations, while the vertical axis represents the coefficient of determination (R^2). The higher the R^2 values, the more accurate the predictions are. As can be seen, for all settings, PaloBoost substantially outperforms the other models even on this extremely difficult problem. The other SGTB implementations overfit so much to the training data and hardly produce better predictions than predicting the mean value ($R^2 \sim 0$). Moreover, PaloBoost achieves consistent results across a range of hyperparameters. This is noteworthy when compared to the drastic differences exhibited by the other two baseline models.

Mortality in ICU. Using the same dataset constructed from the previous experiment, we predict in-hospital mortality of the patients (the original task in the Physionet 2012 challenge). This task also has a substantial impact on resource and management optimization. Hospital staffs can assign highly trained and valuable specialists to specifically cater to high-risk patients' needs, thereby increasing the chance of survival of patients.

Figure 10 shows the predictive performance from the original data (i.e. without label noise), measured in the Area Under the Receiver Operating Characteristics Curve (AUROC). As can be seen, PaloBoost shows consistent and robust performance over different hyperparameter configurations. Interestingly, we did not observe much of drastic performance degradation from other algorithms, though their maximum performances were still below those of PaloBoost. From an overfitting perspective, this suggests the mortality prediction task is an easier problem than predict-

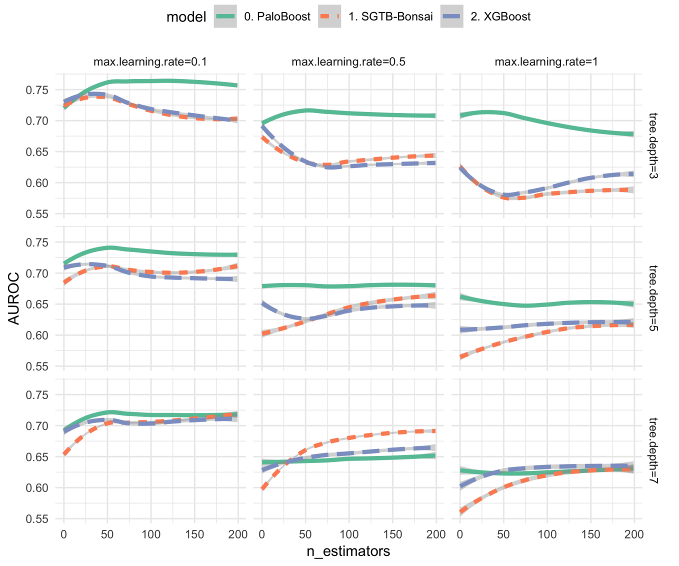


Fig. 11. [Mortality with Label Noise] Predictive performance over boosting iterations from the Mortality Prediction Task with Label Noise. The label noise make the other SGTB implementations easily overfit, while PaloBoost maintains its best performance without much of overfitting.

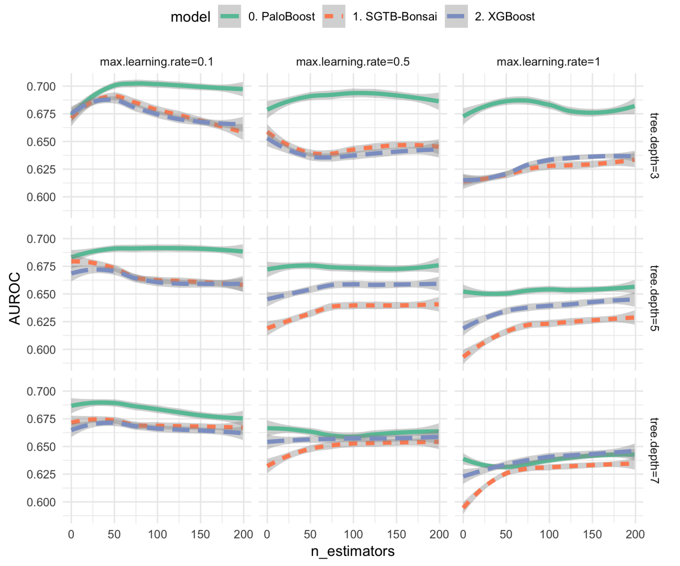


Fig. 12. [Cardiac Arrest] Predictive performance over boosting iterations from the Cardiac Arrest Prediction Task.

ing the length of stay. In fact, overfitting in classification tasks is less prominent than in regression tasks, due to the overfit-resistance nature of the loss function [44].

To better highlight the advantages of PaloBoost, we designed an additional experiment for the mortality task. In healthcare, many data sources are generated by humans, which can lead to various record errors. Such record errors can create both noisy features and noisy labels, making models overfit to such noisy features and labels. In this experiment, we assume random 20% of the labels in the training data have wrong labels, while the test labels are all correct. Figure 11 shows the predictive performance from the noisy label data. We can observe that the AUCs are less than the ones from the original data due to the noisy labels. Moreover, the overfitting behaviors of the other SGTB implementations are more noticeable than the results from the original data. It is noteworthy that, even with this high-level of label noise, PaloBoost shows very consistent predictive performance without much of overfitting.

Cardiac Arrest. Predicting and preventing cardiac arrest is one of the biggest challenges of contemporary cardiology. With accurate predictions of cardiac arrest, the emergency response team can effectively treat patients and increase the chance of survival of patients. For this experiment, we use a dataset from MIMIC-III (Multiparameter Intelligent Monitoring in Intensive Care) database [37]. ICU patients between the ages of 50-75 at the time of admission were extracted. Among those, we focus on the following sets of patients:

1) Cardiac arrest patients who had either asystole or ventricular tachycardia event. The index time for these patients is the time of the first event that is recorded in the event table.
2) Non-cardiac arrest patients who did not experience any cardiac arrest event. The index time for these patients is randomly sampled from their hospital stay.

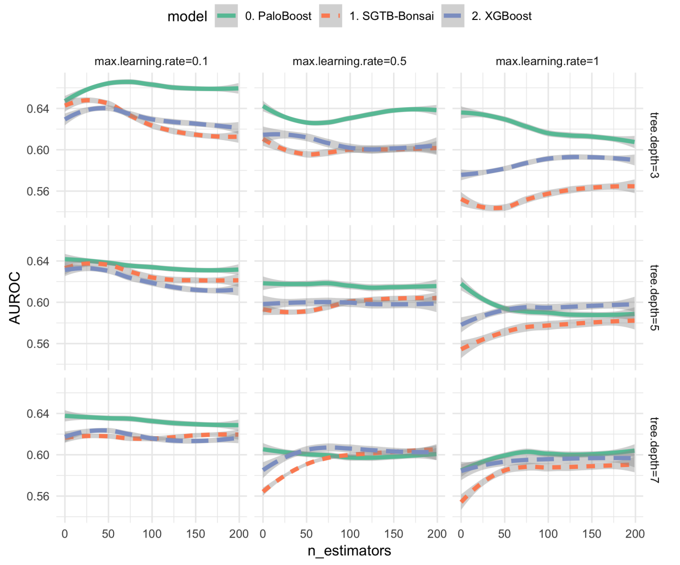


Fig. 13. [Cardiac Arrest with Label Noise] Predictive performance over boosting iterations from the Cardiac Arrest Prediction Task with Label Noise.

We identified 1081 patients that meet our criteria above. The features are constructed from six commonly observed clinical measurements, one derived measurement prior to the index time, and three basic demographic variables for total 11 variables. The measurements include temperature, peripheral capillary oxygen saturation, heart rate, respiratory rate, diastolic blood pressure, systolic blood pressure, and pulse pressure index (the difference between systolic and diastolic blood pressure over the systolic pressure). Our target variable is whether the patient will experience cardiac arrest in the next 6 hours using measurements from the previous 24 hours. For more information about the construction of the dataset, please refer to [39].

Figure 12 illustrates the results of predicting the cardiac arrest of patients. As can be seen, PaloBoost consistently outperforms the other algorithms for all hyperparameter settings. Furthermore, PaloBoost demonstrates stable predic-

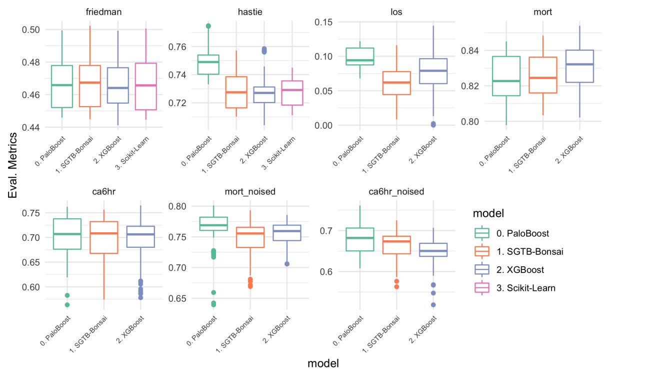


Fig. 14. Predictive Performance of the best-performing hyperparameter configuration for each model and dataset.

tive performance regardless of different boosting iterations and learning rates, while other implementations show some level of performance degradation for certain configurations.

We also present the results with noisy labels, shown in Figure 13. We randomly flipped 20% of the labels in the training data, and trained PaloBoost and the benchmark algorithms. The results are consistent with the other experiments. PaloBoost shows stable predictive performances over different hyperparameter settings and even with many boosting iterations without overfitting.

PaloBoost is originally developed to make a new SGTB algorithm that is less sensitive to hyperparameter configurations. However, as a pleasant side-effect of such an attempt, we find PaloBoost can improve predictive performance for noisy label datasets. Figure 14 shows the predictive performance of the best-performing hyperparameter configuration for each model and dataset. As can be seen, the predictive performance of PaloBoost is comparable to the existing SGTB algorithms. Noticeably, for those two noisy label datasets – "ca6hr_noised" and "mort_noised", PaloBoost improved the predictive performance by meaningful margins. This is because PaloBoost can select statistically meaningful nodes and learning rates using GAP and ALR, removing spurious patterns in training data.

5 Discussions

We introduced PaloBoost, an extension of SGTB, to mitigate overfitting in boosting applications for healthcare data. PaloBoost uses two regularization techniques: gradient-aware pruning and adaptive learning rate estimation. Rather than viewing OOB samples as a third-party observer for tracking errors and feature importances, PaloBoost considers these samples as the second batch of training samples. Based on this new perspective of the OOB samples, PaloBoost dynamically adjusts the tree depths and learning rates at each stage to minimize overfitting by knowing when it needs to "slow down". We showed that our two regularization mechanisms adaptively adjusts to yield lower variance trees and optimal region-specific learning rates. The empirical results demonstrate considerably less sensitivity to the hyperparameter settings – different learning rates and the number of iterations yield comparable predictive performance. While our empirical results suggest that both GAP and ALR are needed to produce hyperparameter-robust models, deeper studies are required to better understand how these two regularizations affect the performance.

An interesting observation drawn from the experiments was the presence of trees with negligible ($\nu \sim 0$) learning rates in PaloBoost. Given that these trees have minimal impact on the overall performance, they can be potentially removed. This should yield a more compact-sized tree boosting model while offering similar predictive performance. Perhaps this can be further extended to encompass a two-steps-forward-one-step-back strategy. By integrating tree removal directly into the algorithm, a more cohesive and compact model can be learned without introducing significant computational overhead. These intriguing ideas are left for future work.

Our extensive experiments confirm that PaloBoost produces robust predictive performance over various real-world healthcare data. It is because healthcare data often exhibit 1) noisy labels, 2) complex data types, 3) missing data, 4) small sample sizes, and 5) a large number of features. Although the experiments presented in this work are limited to healthcare data, extensive studies with other datasets also demonstrated similar results [34].

However, if a dataset has accurate labels and has easy-to-predict target variables, PaloBoost's adaptive learning and pruning may slow down its learning speed. In our experiments with non-healthcare datasets [34], we noticed that PaloBoost sometimes requires more iterations to achieve their best performance. This pattern was more obvious when other SGTB algorithms achieve their best performances easily and do not show overfitting behaviors even with many iterations. Thus, we think that PaloBoost can be best used when traditional SGTB algorithms failed to work well and show substantial overfitting behaviors.

PaloBoost is developed to make SGTB more robust and stable when applying boosting techniques for healthcare data. While significantly less sensitive to the hyperparameters compared to the other implementations, PaloBoost still requires similar hyperparameters ("max" learning rate and tree depth). We note that the ideas presented in this paper are the initial steps toward automating the tuning of SGTB models. Not only can PaloBoost save computation resources and researchers' time, but it can also help democratize SGTB to a wider audience.

References

[1] C. H. Lee and H.-J. Yoon, "Medical big data: promise and challenges," *Kidney Res Clin Pract*, vol. 36, no. 1, pp. 3 – 11, 2017.
[2] L. Nie, M. Wang, L. Zhang, S. Yan, B. Zhang, and T.-S. Chua, "Disease inference from health-related questions via sparse deep learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 8, pp. 2107–2119, Aug 2015.
[3] L. Nie, Y.-L. Zhao, M. Akbari, J. Shen, and T.-S. Chua, "Bridging the vocabulary gap between health seekers and healthcare knowledge," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 2, pp. 396–409, June 2015.
[4] L. Nie, L. Zhang, Y. Yang, M. Wang, R. Hong, and T.-S. Chua, "Beyond doctors: Future health prediction from multimedia and multimodal observations," in *Proceedings of the 23rd ACM International Conference on Multimedia*, ser. MM '15. New York, NY, USA: ACM, 2015, pp. 591–600. [Online]. Available: http://doi.acm.org/10.1145/2733373.2806217
[5] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *The Annals of Statistics*, vol. 29, no. 5, pp. 1189–1232, Oct. 2001.

[6] ——, "Stochastic gradient boosting," *Computational Statistics & Data Analysis*, vol. 38, no. 4, pp. 367–378, Feb. 2002.
[7] L. Breiman, *Classification and regression trees*. Routledge, 2017.
[8] T. Hastie, R. Tibshirani, and J. Friedman, *The elements of statistical learning: data mining, inference, and prediction*, 2nd ed. Springer, 2009.
[9] K. J. Barriga, R. F. Hamman, S. Hoag, J. A. Marshall, and S. M. Shetterly, "Population screening for glucose intolerant subjects using decision tree analyses," *Diabetes Research and Clinical Practice*, vol. 34, pp. S17–S29, 1996.
[10] K. E. Goodman, J. Lessler, S. E. Cosgrove, A. D. Harris, E. Lautenbach, J. H. Han, A. M. Milstone, C. J. Massey, and P. D. Tamma, "A clinical decision tree to predict whether a bacteremic patient is infected with an extended-spectrum β-lactamase–producing organism," *Clinical Infectious Diseases*, vol. 63, no. 7, pp. 896–903, 2016.
[11] R. Bekkerman, "The present and the future of the kdd cup competition," https://www.kdnuggets.com/2015/08/kdd-cup-present-future.html, August 2015.
[12] X. He, J. Pan, O. Jin, T. Xu, B. Liu, T. Xu, Y. Shi, A. Atallah, R. Herbrich, S. Bowers, and J. Q. Candela, "Practical lessons from predicting clicks on ads at facebook," in *Proceedings of the Eighth International Workshop on Data Mining for Online Advertising*, August 2014, pp. 1–9.
[13] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proc. of the 22nd ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, 2016, pp. 785–794.
[14] C. J. Burges, "From ranknet to lambdarank to lambdamart: An overview," Microsoft Corporation, Tech. Rep., 2010. [Online]. Available: https://www.microsoft.com/en-us/research/publication/from-ranknet-to-lambdarank-to-lambdamart-an-overview/
[15] P. Li, C. J. Burges, and Q. Wu, "Learning to rank using classification and gradient boosting," in *Advances in Neural Information Processing Systems*, January 2008.
[16] H. Zhang and V. M. Patel, "Densely connected pyramid dehazing network," in *CVPR*, June 2018, pp. 3194–3203.
[17] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio, "Fitnets: Hints for thin deep nets," https://arxiv.org/abs/1412.6550, March 2015.
[18] E. Barshan and P. Fieguth, "Stage-wise training: An improved feature learning strategy for deep models," in *Proc. of the 1st Int. Workshop on Feature Extraction: Modern Questions and Challenges at NIPS 2015*, 2015.
[19] B. Efron and G. Gong, "A leisurely look at the bootstrap, the jackknife, and cross-validation," *The American Statistician*, vol. 37, no. 1, pp. 36–48, February 1983.
[20] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *Proc. of the 14th Int. Joint Conf. on Artificial intelligence*, vol. 2, August 1995, pp. 1137–1143.
[21] G. Ridgeway, "Generalized boosted models: A guide to the gbm package," https://cran.r-project.org/web/packages/gbm/vignettes/gbm.pdf, 2012.
[22] W. Jiang, "On weak base hypotheses and their implications for boosting regression and classification," *The Annals of Statistics*, vol. 30, no. 1, pp. 51–73, November 2002.
[23] M. Hardt and A. Blum, "The ladder: A reliable leaderboard for machine learning competitions," in *Proceedings of the 32nd International Conference on Machine Learning*, 2015.
[24] B. Recht, R. Roelofs, L. Schmidt, and V. Shankar, "Do cifar-10 classifiers generalize to cifar-10?" https://arxiv.org/abs/1806.00451, June 2018.
[25] L. Breiman, "Bagging predictors," *Machine Learning*, vol. 24, no. 2, pp. 123–140, 1996.
[26] ——, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.
[27] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
[28] J. H. Friedman, "Multivariate adaptive regression splines," *The Annals of Statistics*, vol. 19, no. 1, pp. 1–67, March 1991.
[29] L. Breiman, "Arcing classifier (with discussion and a rejoinder by the author)," *The Annals of Statistics*, vol. 26, no. 3, pp. 801–849, Jun. 1998.
[30] H. Schwenk and Y. Bengio, "Boosting neural networks," *Neural Computation*, vol. 12, no. 8, pp. 1869–1887, August 2000.
[31] Y. Freund and R. E. Schapire, "A short introduction to boosting," *Journal of Japanese Society for Artificial Intelligence*, vol. 14, no. 5, pp. 771–780, September 1999.
[32] S. S. a Minitab company, "Treenet - gradient boosting," https://www.salford-systems.com/products/treenet, 2018.
[33] J. Elith, J. R. Leathwick, and T. Hastie, "A working guide to boosted regression trees," *Journal of Animal Ecology*, vol. 77, no. 4, pp. 802–813, July 2008.
[34] Y. Park and J. C. Ho, "Paloboost: An overfitting-robust treeboost with out-of-bag sample regularization techniques," https://arxiv.org/abs/1807.08383, July 2018.
[35] Y. Park, "Bonsai-dt - programmable decision tree framework," https://yubin-park.github.io/bonsai-dt/.
[36] A. L. Goldberger, L. A. N. Amaral, L. Glass, J. M. Hausdorff, P. C. Ivanov, R. G. Mark, J. E. Mietus, G. B. Moody, C.-K. Peng, and H. E. Stanley, "PhysioBank, PhysioToolkit, and PhysioNet: Components of a new research resource for complex physiologic signals," *Circulation*, vol. 101, no. 23, pp. e215–e220, 2000.
[37] A. E. Johnson, T. J. Pollard, L. Shen, H. L. Li-wei, M. Feng, M. Ghassemi, B. Moody, P. Szolovits, L. A. Celi, and R. G. Mark, "Mimic-iii, a freely accessible critical care database," *Scientific data*, vol. 3, p. 160035, 2016.
[38] M. Sotoodeh and J. C. Ho, "Improving length of stay prediction using a hidden markov model," in *AMIA Joint Summits on Translational Science*, 2019.
[39] J. C. Ho and Y. Park, "Learning from different perspectives: Robust cardiac arrest prediction via temporal transfer learning," in *Proc. of the 39th Annual Int. Conf. of the IEEE Engineering in Medicine and Biology Society*, 2017.
[40] L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin, "CatBoost: unbiased boosting with categorical features," *arXiv:1706.09516 [cs.LG]*, Jun. 2017.
[41] A. V. Dorogush, V. Ershov, and A. Gulin, "CatBoost: gradient boosting with categorical features support," *Workshop on ML Systems at NIPS 2017*, 2017.
[42] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "LightGBM: A highly efficient gradient boosting decision tree," in *Advances in Neural Information Processing Systems 30*, 2017, pp. 3146–3154.
[43] J. H. Friedman, E. Grosse, and W. Stuetzle, "Multidimensional additive spline approximation," *SIAM Journal on Scientific and Statistical Computing*, vol. 4, no. 2, pp. 291–301, 1983.
[44] J. Friedman, T. Hastie, and R. Tibshirani, "Additive logistic regression: A statistical view of boosting: Rejoinder," *The Annals of Statistics*, vol. 28, no. 2, pp. 400–407, April 2000.

Yubin Park is Principal at Bonsai Research, LLC. He received his Ph.D. and M.S.E in electrical and computer engineering from the University of Texas at Austin and B.S. from KAIST in 2014, 2011 and 2008 respectively. He co-founded a healthcare analytics start-up (Accordion Health), and served the CEO and CTO roles until the company was sold to Evolent Health, Inc.

Joyce C. Ho is an assistant professor in the Computer Science Department at Emory University. Her research is in data mining and machine learning, focusing on healthcare applications. She received her Ph.D. from the University of Texas at Austin, M.S. and B.S. from Massachusetts Institute of Technology. She co-founded a successful healthcare analytics company (Accordion Health) and previously worked at Lawrence Livermore National Laboratory.