PaloBoost: An Overfitting-robust TreeBoost with Out-of-Bag Sample Regularization Techniques

YUBIN PARK and JOYCE C. HO, Emory University, USA

Stochastic Gradient TreeBoost is often found in many winning solutions in public data science challenges. Unfortunately, the best performance requires extensive parameter 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 parameter settings. PaloBoost uses the under-utilized 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. We illustrate how these regularization techniques can be efficiently implemented and propose a new formula for calculating feature importance to reflect the node coverages and learning rates. Extensive experimental results on seven datasets demonstrate that PaloBoost is robust to overfitting, is less sensitivity to the parameters, and can also effectively identify meaningful features.

CCS Concepts: • **Computing methodologies** → **Boosting**; **Regularization**; *Supervised learning by classification*; *Supervised learning by regression*;

Additional Key Words and Phrases: Machine learning, Boosting, Statistical learning, Predictive models, Supervised learning

ACM Reference Format:
Yubin Park and Joyce C. Ho. 2018. PaloBoost: An Overfitting-robust TreeBoost with Out-of-Bag Sample Regularization Techniques. 1, 1 (July 2018), 21 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Stochastic Gradient TreeBoost (SGTB) is one of the most widely used off-the-shelf machine learning algorithms [4, 11, 20, 24]. SGTB is a stage-wise additive model where the base trees are fitted to the subsampled gradients (or errors) at each stage [18, 19]. The randomness introduced by subsampling speeds up the computation time and mitigates overfitting. Consequently, SGTB can provide robust performance over various classification, regression, and even ranking tasks [8, 29]. 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 [11].

While SGTB can generally provide reasonable performance with the default parameter settings, to achieve its best performance usually requires extensive parameter tuning. The four parameters in SGTB are maximum depth of the tree, number of trees, learning rate, and the subsampling rate. While the maximum depth, subsampling rate, and learning rate have general guidelines, each parameter is not independent of one another. Having deeper trees and more trees can reduce training errors more rapidly, while potentially increasing the chance of overfitting [18, 37]. Lower learning rates can mitigate overfitting to the training data at each stage, but more trees are

Authors' address: Yubin Park, yubin.park@gmail.com; Joyce C. Ho, Emory University, Mathematics and Science Center, W414, Atlanta, GA, 30322, USA, joyce.c.ho@emory.edu.

XXXX-XXXX/2018/7-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

needed to reach a similar level of performance [37]. Thus, the delicate balancing act between overfitting and best performance is more of an art than a science in practice.

Exhaustive parameter tuning can be a bottleneck for many time-sensitive or performance-critical practical applications. An ideal boosting algorithm should automatically learn what the four parameter values should be to achieve its best performance without overfitting. Moreover, a smaller number of trees is equivalent to faster convergence, reduced probability of overfitting, and better performance. This suggests two improvements: (1) the learning rate should be higher when the model is further away from convergence, and lower when the model is approaching its best performance and (2) the depth of the tree should change dynamically to account for the complexity of the target. Thus the question is whether such a boosting algorithm can be built.

We present PaloBoost, a boosting algorithm that makes it easier to tune the parameters 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 [5, 7]. 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 [37]. However, OOB samples are under-utilized in SGTB. They merely play an observer role in the overall training process, often ignored and left unused.

PaloBoost, on the other hand, treats OOB samples as a second batch of training samples for adjusting 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 does not decrease, the tree is too specific and likely overfit on the training data. 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 randomly partitions the training samples into two sets: one is used for learning the tree structure, and the other is used to prune and adjust the learning rates. This two-stage training process frequently appears in many machine learning algorithms to estimate two distinct sets of parameters or models [3, 38, 42]. For example, in hyperparameter tuning [14, 28], the parameters are sequentially estimated from two disjoint datasets: (1) regular parameters from a training set and (2) hyperparameters from a validation set. In PaloBoost, this translates to (1) in-bag samples for growing trees and (2) OOB samples for pruning and adjusting learning rates. The main difference lies in the fact that 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 [34] and XGBoost [11]. Our benchmark datasets include one simulated data [17] and six real datasets from Kaggle [26] and UCI repositories [12]. The empirical results demonstrate stable, predictive performance in the presence of noisy features. PaloBoost is considerably less sensitive to the hyperparameters and hardly suffers significant performance degradations with more trees. The results also illustrate the adaptive learning rates and tree depths that guard against overfitting. Moreover, we demonstrate the potential of our proposed feature importance formula to provide better feature selection than other SGTB implementations. Thus, PaloBoost offers a robust, SGTB model that can save computation resources and researcher's time, 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 [6]. 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 [18]. 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 [18] 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 [39], support vector machine [40], and regression tree [18]. Algorithm 1 outlines the details of GBM.

ALGORITHM 1: Gradient Boosting Machine with Base Learner $h(\mathbf{x}; \mathbf{a})$

$F_0 = \arg\min_\beta \sum_{i=1}^{N} L(y_i, \beta)$
for $m \leftarrow 1$ ***to*** M **do**
  $z_i = -\frac{\partial}{\partial F(\mathbf{x}_i)} L(y_i, F(\mathbf{x}_i))\Big|_{F=F_{m-1}}$, for $i = 1, \cdots, N$
  $\mathbf{a}_m = \arg\min_{\mathbf{a}} \sum_{i=1}^{N} \|z_i - h(\mathbf{x}_i; \mathbf{a})\|^2$
  $\beta_m = \arg\min_\beta \sum_{i=1}^{N} L(y_i, F_{m-1}(\mathbf{x}_i) + \beta h(\mathbf{x}_i; \mathbf{a}_m))$
  $F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \beta_m h(\mathbf{x}; \mathbf{a}_m)$

An alternative perspective of GBM is to view it as an ensemble where the approximation function, $h(\mathbf{x})$, is the base learner and β_m represents the corresponding weight. The GBM model (outlined in Algorithm 1) produces a final output F of the form:

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

In fact, XGBoost, a variant of GBM, uses this approach to develop a scalable and flexible boosting system [11]. By assuming that F takes the ensemble form, the optimal base learners are derived by applying the Taylor approximation and a greedy optimization technique. PaloBoost is better understood in the context of the original GBM work [18]. Thus, the notations and formulation will follow Friedman's work.

2.2 Stochastic Gradient TreeBoost

Stochastic Gradient TreeBoost (SGTB) introduces two important modifications to GBM: (1) tree structure-aware step sizes and (2) subsampling at each stage [18]. 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 [5] and AdaBoost [16]. 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.

However, subsampling alone may be insufficient. 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 would need more iterations, but may achieve better generalization [19]. Algorithm 2 illustrates the details of SGTB with a learning rate ν. Thus, SGTB has four different parameters that require tuning: (1) tree size (related to tree depth) (J); (2) number of trees (M); (3) the learning rate (ν); and (4) the subsampling rate (q).

ALGORITHM 2: Stochastic Treeboost with Learning Rate (ν)

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

SGTB is widely available in many easy-to-use open-source packages. Some implementations include Generalized Boosted Models available in `R` [37], TreeNet by Salford Systems [1], Gradient Boosting in Scikit-Learn [34], and XGBoost implemented in `C++` [11]. Although all packages follow Algorithm 2, the implementations 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 [18] by default, and XGBoost uses a custom regularized splitting criterion that is obtained by greedily minimizing the Taylor approximation of the loss function [11]. 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. However, there is no conclusive evidence that one implementation is the best, as the performance differences are dataset-dependent.

3 PALOBOOST

The performance of SGTB is dependent on its hyperparameters: tree depth, number of trees, subsampling rate, and learning rate. Unfortunately, each parameter is not independent of the others and makes the tuning process quite difficult. As an example, the ν-M trade-off [18, 19], 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. Moreover, our implementation provides comparable training times to existing state-of-the-art SGTB implementations.

PaloBoost mitigates overfitting via adaptive learning rates and tree depths. The main idea centers around the *under-utilized* out-of-bag (OOB) samples to tune these parameters. We observed that *both validation and OOB samples* are not seen during the training phase. Thus, the OOB samples can be used to estimate the learning rate and the tree depth at each individual stage. The sample principle can be applied at each stage even though the OOB samples are different as the learning rate and tree depth are tree-specific parameters. 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. Figure 1 illustrates the conceptual differences in the learning process for multiple-stages SGTB and PaloBoost.

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

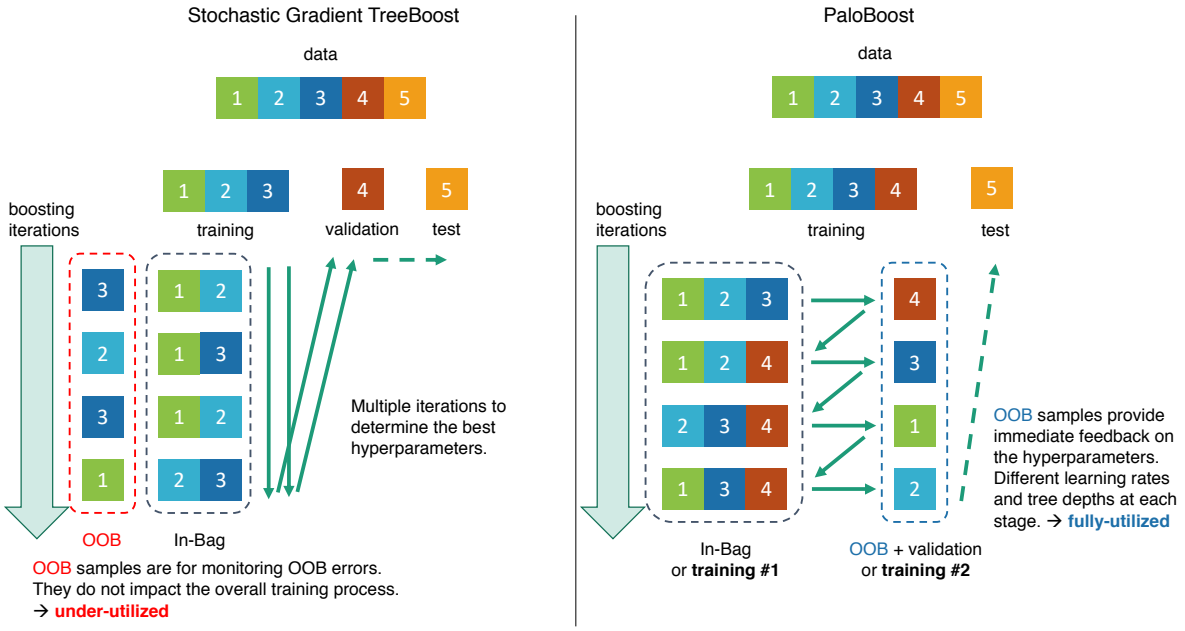


Fig. 1. Data flows in SGTB and PaloBoost.

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 parameter [11], our empirical studies showed little impact unless the default parameter 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 parameter 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. Therefore, a new pruning approach is necessary to account for these aspects.

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. 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:

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

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 3.

ALGORITHM 3: Gradient-Aware Pruning

$\{(R_j, R_k)\}$ = 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 $\sum_i^{L_j} Loss(y_i, \hat{y}_i) < \sum_i^{L_j} Loss(y_i, \hat{y}_i + \nu_{max}\gamma_j)$ **then**
    do_merge = True
  else if $\sum_i^{L_k} Loss(y_i, \hat{y}_i) < \sum_i^{L_k} Loss(y_i, \hat{y}_i + \nu_{max}\gamma_k)$ **then**
    do_merge = True
  else
    do_merge = False
  if *do_merge* **then**
    Merge(R_j, R_k)

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 [23]. 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. 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:

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

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

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 parameter v_{max}. As a result, the effective region-specific learning rate ranges between zero and the maximum specified learning rate. Algorithm 4 illustrates our adaptive learning rate strategy.

ALGORITHM 4: Adaptive Learning Rate 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**
    $v_j = \text{clip}\left(\frac{\sum_{i=1}^{L_j}(y_i - \hat{y}_i)}{\gamma_j L_j}, 0, v_{max}\right)$
  else if *distribution==“bernoulli“* **then**
    $v_j = \text{clip}\left(\log\left(\frac{\sum_i^{L_j} y_i}{\sum_i^{L_j}(1-y_i)\exp(\hat{y}_i)}\right)/\gamma_j, 0, v_{max}\right)$

Gradient-aware pruning 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. Algorithm 5 provides the details of PaloBoost. As learning rate and tree depth values are re-estimated during the training process, there is less sensitivity to the maximum learning rate parameter v_{max} and tree depth. Moreover, these modifications also provide a safeguard for a variety of subsampling rates. Therefore, only the number of trees, M, requires “extensive” tuning.

3.3 Modified Feature Importance

Feature importance in SGTB is calculated as the summation of squared error improvements by the feature [15]. Although the formula is adopted from RandomForest [7], the measure is purely based on heuristic arguments [18]. While the feature importance has proven useful in many applications, we notice that ignores two important aspects of SGTB: the leaf estimates and the coverage of the region. As seen in the SGTB algorithm (Algorithm 2), 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,

ALGORITHM 5: PaloBoost

$F_0 = \arg\min_\beta \sum_{i=1}^{N} L(y_i, \beta)$
for $m \leftarrow 1$ ***to*** M **do**
  $\{y_i, \mathbf{x}_i\}_1^{N'} = \text{Subsample}(\{y_i, \mathbf{x}_i\}_1^N, \text{rate} = q)$
  $z_i = -\frac{\partial}{\partial F(\mathbf{x}_i)} L(y_i, F(\mathbf{x}_i))\Big|_{F=F_{m-1}}$, for $i = 1, \cdots, N'$
  $\{R_{jm}\}_1^J = \text{RegressionTree}(\{z_i, \mathbf{x}_i\}_i^{N'})$
  $\gamma_{jm} = \arg\min_\gamma \sum_{\mathbf{x}_i \in R_{jm}} L(y_i, F_{m-1}(\mathbf{x}_i) + \gamma)$, for $j = 1, \cdots, J$
  $\{R_{jm}, \gamma_{jm}\}_1^{J'} = \text{Gradient-Aware-Pruning}(\{R_{jm}, \gamma_{jm}\}_1^J, \nu_{max})$
  $\nu_{jm} = \text{Learning-Rate-Adjustment}(\gamma_{jm}, \nu_{max})$, for $j = 1, \cdots, J'$
  $F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \sum_{j=1}^{J} \nu_{jm}\gamma_{jm}\mathbb{1}(\mathbf{x} \in R_{jm})$

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 very small, the feature importance contribution is minimal.

4 EXPERIMENTAL RESULTS

PaloBoost is implemented in Python using the Bonsai Decision Tree framework [33], 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[1].

PaloBoost is evaluated on a variety of datasets (simulated and real-world) and compared against state-of-the-art 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 the specified hyperparameters?
- How does the predictive performance compare with other SGTB implementations?
- Is PaloBoost more computationally expensive than existing SGTB implementations?

All experiments are carried out on a single machine running macOS High Sierra 10.13.6 with 2.7 GHz Intel Core i5 CPU and 8 GB of RAM.

4.1 Datasets

Seven different datasets are used to evaluate the SGTB variants, including PaloBoost. We use one simulated dataset and six publicly available datasets, two from the UC Irvine (UCI) machine learning repository [12] and four from Kaggle [26], a popular data science competition platform. Our benchmark study consists of three regression tasks and four classification tasks. The sizes of the datasets range from a few thousand samples (1,994

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

Table 1. Benchmark Datasets. For regression tasks, we list the standard deviation of the target variable, whereas the class ratios for classification tasks. The "# Real" and "# Cat" columns represent the number of numeric and categorical features, respectively. The "# Final" column shows the final number of features after our pre-processing step.

Dataset	Task	Missing Data	Target Stats	# Samples	# Real	# Cat	# Final
Friedman-Sim [17]	REG	None	$\sigma(y) = 6.953$	10,000	10	0	10
Mercedes-Benz [32]	REG	None	$\sigma(y) = 12.67$	4,209	368	8	460
Crime-Rate [36]	REG	Present	$\sigma(y) = 0.233$	1,994	126	1	127
Amazon-Empl [2]	CLS	None	$\mu(y) = 0.942$	32,769	0	9	115
Pulsar-Detect [30, 31]	CLS	None	$\mu(y) = 0.092$	17,898	8	0	8
Carvana [10]	CLS	Present	$\mu(y) = 0.123$	72,983	17	14	151
BNP-Paribas [9]	CLS	Present	$\mu(y) = 0.761$	114,321	112	19	273

samples) to a little bit over a hundred thousand samples (114,321 samples). Small datasets are included to highlight the performance sensitivity with respect to the hyperparameters. Table 1 lists the seven real-world datasets and their associated characteristics. A brief overview of each dataset will be provided in the context of their respective tasks.

Many of the datasets contain both numeric and categorical features. For the categorical features, the top 20 frequent categories are selected and transformed using one-hot encoding (also known as dummy-coding). To minimize the impact of preprocessing, no additional techniques were applied (e.g. imputation, outlier detection). 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.

4.2 Baselines

We compare PaloBoost with three other baseline models as follows:

- Scikit-Learn [34], the de-facto machine learning library in Python that has been widely adopted.
- XGBoost [11], the battle-tested library that has won many data science challenges.
- Bonsai-SGTB, an SGTB implementation using the Bonsai framework [2].

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 are similar to the 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 [13, 35]. LightGBM primarily focuses on training speed using less memory in distributed settings [27]. 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 a 30/70 train-test ratio (i.e., 30% for training and 70% for test). While we performed extensive experiments using other ratio choices and found similar behaviors, this specific choice showcases the sensitivity of SGTB to hyperparameters. With larger training ratios (e.g., 50/50, 60/40, 70/30, 80/20), we found that it was difficult to visualize the overfitting behaviors for many of the baseline models. On the contrary, for smaller

[2]https://yubin-park.github.io/bonsai-dt/

training ratios such as 10/90 and 20/80, the results were significantly more favorable to PaloBoost. The 30/70 split best illustrated the characteristics of PaloBoost without significantly sacrificing the training sample size.

For each SGTB implementation, we tested three different learning rates that spanned the spectrum for ν (1.0, 0.5, and 0.1). Also for consistency across the models, the tree depth was set to be five levels (note that PaloBoost will have lower depths due to Gradient-aware pruning). Unless otherwise specified, default parameter settings were used for all our benchmark models. The trained models (on the 30% training set) are then applied to the test set where we track the predictive performance. This is measured using the coefficient of determination (R^2) and the Area Under the Receiver Operating Characteristic Curve (AUROC) for the regression and classification tasks, respectively. We also track two different metrics for PaloBoost:

- **Average Learning Rate**: For visualization purposes, we defined the average learning rate per stage to compress the different learning rates for each disjoint region of a tree. This is a weighted average of the learning rates, where the weights are proportional to the leaf coverage, and is defined as:

$$\bar{\nu}_m = \frac{\sum_j \nu_{jm}|R_{jm}|}{\sum_j |R_{jm}|} \tag{15}$$

- **Prune Rate**: We measure the number of nodes removed by the Gradient-aware pruning mechanism. The prune rate is defined as follows:

$$\text{Prune Rate} = \frac{J - J'}{J} \tag{16}$$

  where J and J' represent the number of disjoint regions in a tree before and after Gradient-aware Pruning, respectively.

4.4 Simulated Dataset

Description. The simulated dataset is adopted from Friedman [17, 21], to introduce noisy features (i.e., features with zero importance). The formula for the dataset is as follows:

$$y = 10\sin(\pi x_0 x_1) + 20(x_2 - 0.5)^2 + 10x_3 + 5x_4 + 5\epsilon \tag{17}$$

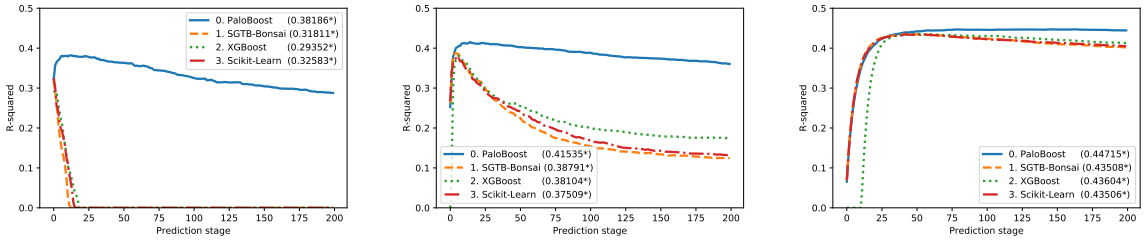
$$\epsilon \sim \text{Normal}(0, 1) \tag{18}$$

$$x_i \sim \text{Uniform}(0, 1) \text{ where } i = 0 \ldots 9 \tag{19}$$

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 used to generate the target (y) in Equation 17. Our simulated dataset contains 10,000 samples.

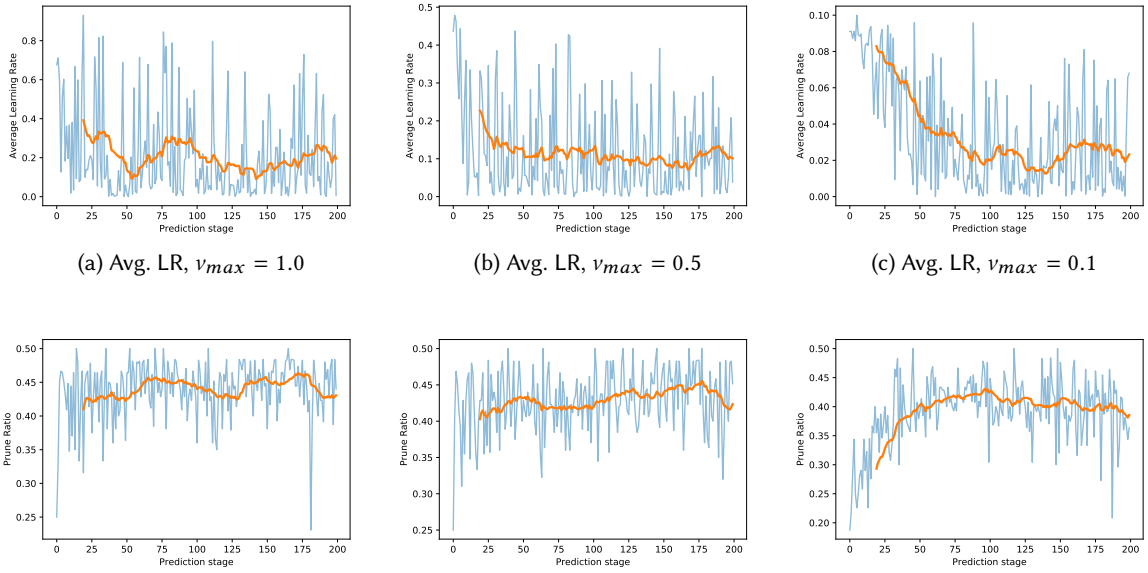
Predictive Performance. Figure 2 shows the coefficient of determination (R^2) for the three different learning rates. PaloBoost exhibits the best and most stable predictive performance for all the learning rates. While the other SGTB implementations show a significant degradation in predictive performance as they iterate more, 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 25 iterations. Moreover, the predictive performance of PaloBoost varies considerably less across the three different learning rates, illustrating better robustness to the parameter setting (ν_{max}).

Average Learning Rate and Prune Rate. The gradual decline of predictive performance in PaloBoost can be better understood by analyzing the prune rate and average learning rate at each stage. Figure 3 displays the average learning rate (top row) and the prune rate (bottom row) over the same iterations. We also overlay the 20-step moving average (colored in orange), to visualize the overall trend. As can be seen, the learning rate for each stage is quite different, and 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 (more noticeable in $\nu_{max} = 0.1$). In



(a) R^2, $\nu_{max} = 1.0$ (b) R^2, $\nu_{max} = 0.5$ (c) R^2, $\nu_{max} = 0.1$

Fig. 2. Predictive performance on the Simulated Dataset.



(a) Avg. LR, $\nu_{max} = 1.0$ (b) Avg. LR, $\nu_{max} = 0.5$ (c) Avg. LR, $\nu_{max} = 0.1$

(d) Prune Rate, $\nu_{max} = 1.0$ (e) Prune Rate, $\nu_{max} = 0.5$ (f) Prune Rate, $\nu_{max} = 0.1$

Fig. 3. The average learning rate and prune rate for PaloBoost on the Simulated Dataset.

addition, the prune rate increases to yield lower variance trees. Thus, the two regularization mechanisms serve as a guard against overfitting.

Computational Speed. Gradient-aware pruning and adaptive learning add computational complexity to SGTB. To determine the impact of our modifications, we measured the training speeds of the four SGTB models (PaloBoost, Bonsai-SGTB, Scikit-learn, and XGBoost). Figure 4 shows the measured training time over simulated training sample sizes. As can be seen, the performance of PaloBoost is comparable with the other implementations even for large sample sizes (e.g. one million samples). While PaloBoost is marginally slower than SGTB-Bonsai, the two regularization techniques do not slow down the overall training process much.

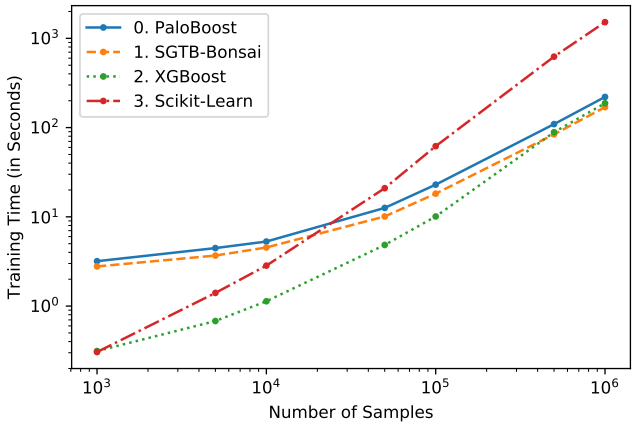


Fig. 4. Runtime experiment on a macOS High Sierra 10.13.6 machine with 2.7 GHz Intel Core i5 CPU, Python 2.7.14, Scikit-Learn 0.19.1, and XGBoost 0.6.

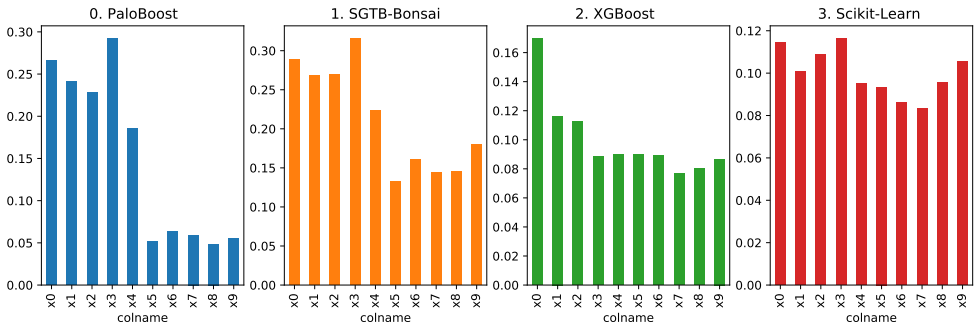


Fig. 5. The feature importances after 200 iterations. Only the first five features, x_0 to x_4, are used to generate the target.

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. We compared the feature importances from the four different trained models with 200 iterations ($M = 200$) and a learning rate of 0.1 ($\nu_{max} = 0.1$). 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 5 shows the feature importances from all four SGTB models[3]. PaloBoost clearly identifies the noisy features ($x_5 - x_9$), while XGBoost and Scikit-Learn estimate similar importances between the relevant and noisy features. Although preliminary, this indicates PaloBoost can be also very useful for the feature selection processes.

The superiority of PaloBoost's feature importance is likely due to two different aspects. First, gradient-aware pruning and adaptive learning rates are removing noisy features, as regions that do not generalize well are either pruned or have a small adaptive learning rate. Secondly, the modified importance formula encapsulates the

[3]The results are similar for $M = 50$ but less dramatic.

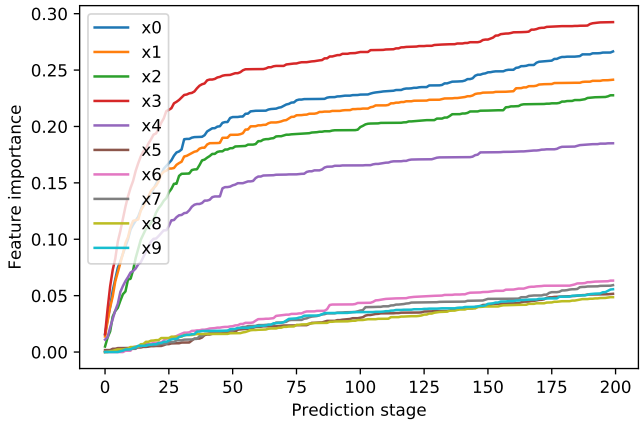


Fig. 6. The evolution of feature importances at each stage of PaloBoost. This plot resembles the coefficient path plot in Elastic Net [43].

learning rate, coverage, and impact of the feature, using the leaf-centric multiplicative factor $\nu_{jm}|R_{jm}||\gamma_{jm}|$. This can be confirmed in Figure 5 by comparing the importance estimates for features $x_5 - x_9$ between SGTB-Bonsai and XGBoost (or Scikit-Learn). SGTB-Bonsai uses the proposed feature importance formula with the same learning rate ($nu_{jm} = 0.1$), yet still identifies lower importance values for $x_5 - x_9$. While accounting for the region size ($|R_{jm}|$) and node estimate (γ_{jm}) is clearly important, the adaptive learning rate provides further separation of the noisy features. Thus, we conclude that both aspects play important roles for the superior performance.

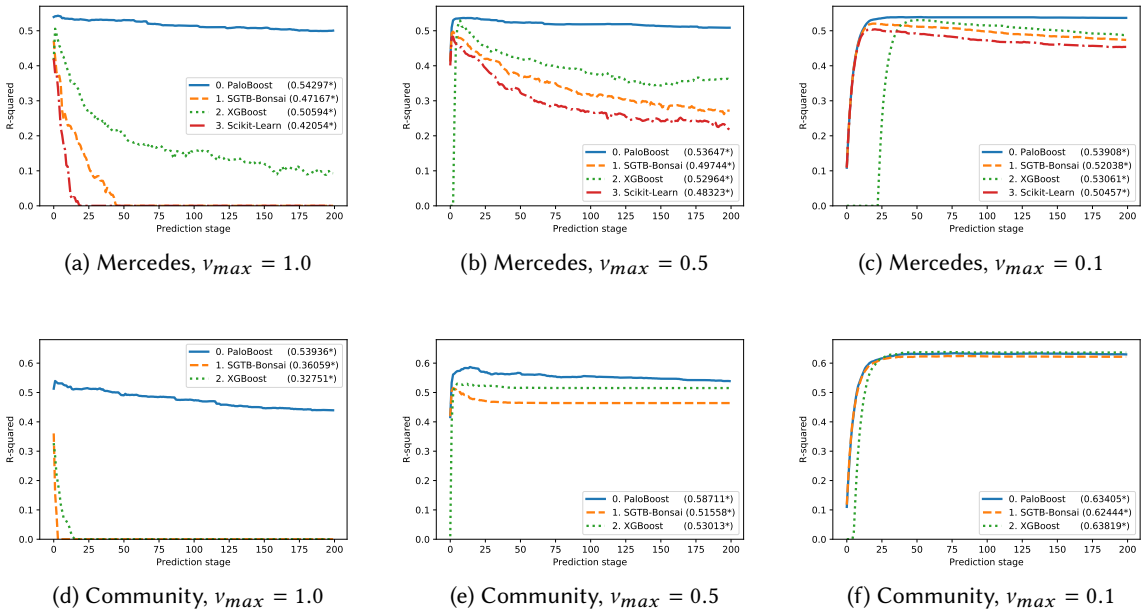
We further explored PaloBoost's feature importance estimates as a function of the iterations for the same learning rate ($\nu_{max} = 0.1$). Figure 6 visualizes the evolution of the feature importance estimates and offers a perspective analogous to the coefficient path in Elastic Net [43]. Using this plot, we can inspect the inner mechanism of PaloBoost more thoroughly. As can be seen, the feature importances for the first five features rapidly increases through the first 30 iterations, after which they slowly increase. Moreover, after 20 iterations, the order of the features no longer change and reflect the true feature rank. We also note that this visualization can also be used to determine the stopping criteria for the number of iterations.

4.5 Regression Tasks

The four SGTB models are evaluated on two publicly available regression datasets, the Mercedes-Benz Greener Manufacturing Dataset and the Communities Crime Dataset. While only the coefficient of determination (R^2) results are presented, the results from our extensive experimental studies on adaptive learning rate and prune rate are available on the PaloBoost package webpage.

4.5.1 Mercedes-Benz Greener Manufacturing Dataset. Daimler, the parent company of Mercedes-Benz, performs exhaustive tests on its pre-release cars for the safety and reliability. The process can easily consume a substantial amount of gas and time, and generate a large volume of carbon dioxide. In the effort to reduce the testing time and conserve the environment, the company released its dataset that contains various measurements (features) and time to finish the test (target) through the Kaggle platform[4]. As summarized in Table 1, the dataset has only

[4]https://www.kaggle.com/c/mercedes-benz-greener-manufacturing



(a) Mercedes, $\nu_{max} = 1.0$ (b) Mercedes, $\nu_{max} = 0.5$ (c) Mercedes, $\nu_{max} = 0.1$

(d) Community, $\nu_{max} = 1.0$ (e) Community, $\nu_{max} = 0.5$ (f) Community, $\nu_{max} = 0.1$

Fig. 7. The predictive performance, measured by R^2, on the regression tasks. The results from Mercedes-Benz dataset are on the top row, and the Community-Crime on the bottom row. The y-axis is aligned for each dataset (row) for ease of comparison across learning rates.

4,209 samples with 460 features (post-processed). Thus, many complex models are likely to overfit on the small training sample and need to properly address the curse of dimensionality [25, 41].

The top row in Figure 7 shows the predictive performance (R^2) on the Mercedes-Benz dataset. Similar to the simulation results (Figure 2), PaloBoost achieves the best and most stable performance. More impressively, PaloBoost suffers minimal degradation at the high learning rate ($\nu_{max} = 1.0$), while a noticeable decay occurs immediately for the other three models. Furthermore, the performance differences across the three learning rates are marginal for PaloBoost, which demonstrates the robustness of the maximum learning rate parameter setting. Unlike the other SGTB implementations that use a fixed learning rate, we observed that the average learning rate in PaloBoost dropped after a few iterations. Moreover, approximately 40% of the regions were pruned during the gradient-aware prune mechanism. Together, these modifications contribute to the robust performance of PaloBoost.

4.5.2 Communities and Crime Dataset. Communities that have similar socioeconomic and law enforcement profiles may also experience similar types of crimes. Such information can be shared across police departments to enable cooperation. In the effort to make such a data-driven system, Redmond [36] created a dataset by combining three data sources: US Census in 1990, US FBI Uniform Crime Report in 1995, and 1990 US Law Enforcement Management and Administrative Statistics Survey. The dataset, available on the UCI Repository[5], contains 127 features that include the percent of unemployed, percent under poverty, median income, etc. and the violent crime rate of the community (numeric target). This dataset contains only 1,994 samples with 127 features and has

[5]http://archive.ics.uci.edu/ml/datasets/communities+and+crime

missing values (see Table 1), therefore Scikit-Learn is omitted as a baseline. The author of the dataset has stated that it has been carefully curated with only features that have a plausible connection to crime.

The bottom row in Figure 7 illustrates the results from the Communities and Crime Dataset experiments. As can be seen, PaloBoost exhibits the most stable behavior across many iterations. However, unlike the previous two datasets (Mercedes-Benz and simulated), the performance of the benchmark models do not suffer a significant degradation. Since the feature selection has already been done by the author, the features are not noisy, thereby mitigating the curse of dimensionality. While PaloBoost does not obtain the highest R^2 overall compared to XGBoost, we note that the values between the different learning rates are much smaller in comparison. This eliminates the need to finely tune the learning rate and the number of iterations.

4.6 Classification Tasks

Next, we present the various SGTB models on four different classification tasks: Amazon Employee Access, Pulsar Detection, Carvana, and BNP-Paribas. Although we only present the predictive performance (AUROC), the adaptive learning rate and prune rate results are available on the PaloBoost package webpage.

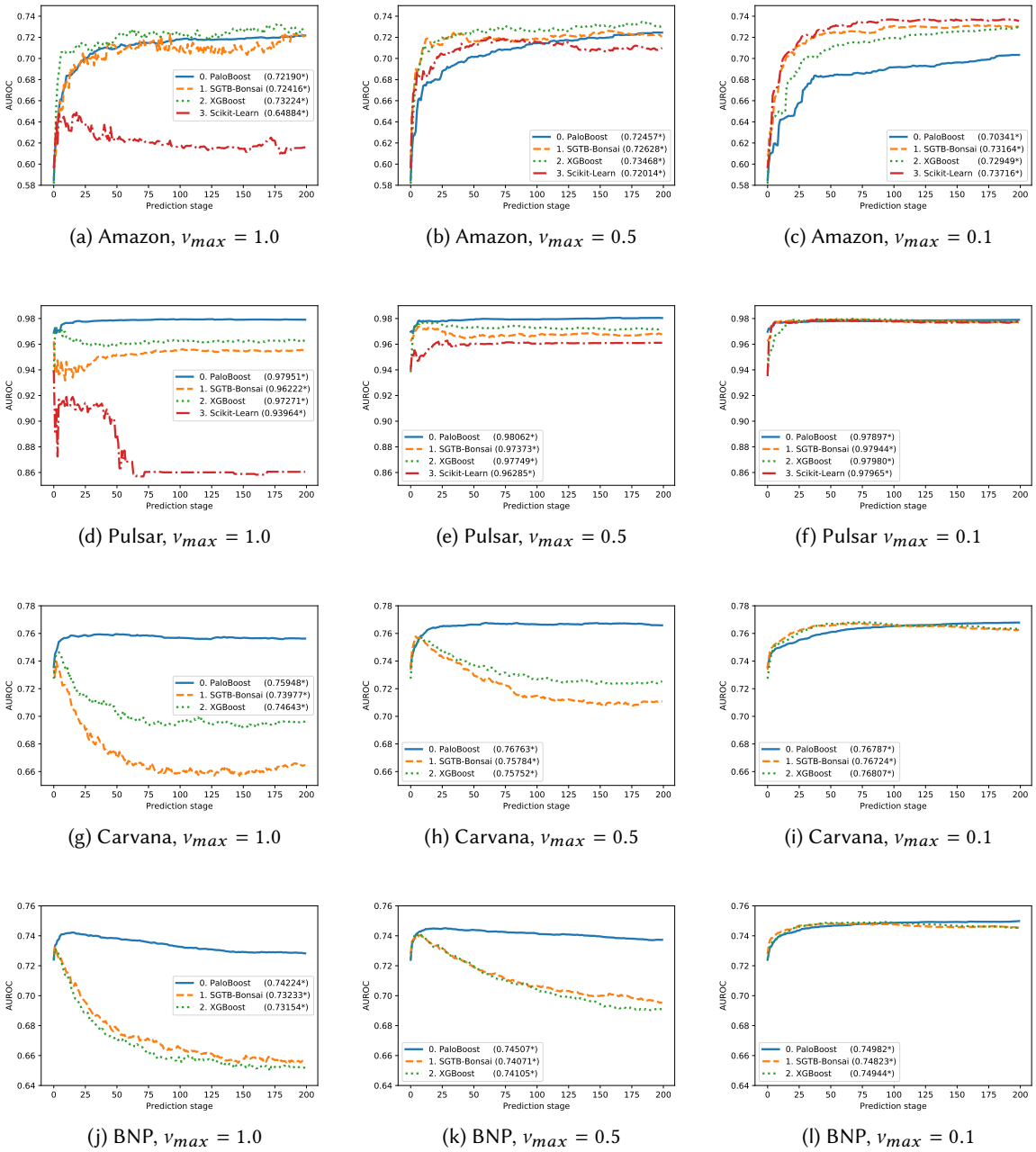
4.6.1 Amazon Employee Access Dataset. Depending on their roles, Amazon's employees have different access rights on their internal web resources. Supervisors often spend a considerable amount of time manually granting and revoking the access rights of their employees. To automate this process and increase the overall work efficiency, Amazon released its historical employee access dataset on Kaggle[6] that contains employee attributes, resource ID, and whether the access was granted or not (target). Interestingly, the dataset only contains eight categorical features (preprocessed to 115 features) for 32,769 employees (see Table 1). Also, many top-scoring solutions on the Kaggle leaderboard noted the importance of capturing the interaction between categorical features to improve the predictive performance. This is due to the fact that the employees' access rights are somewhat well defined given a specific configuration of employee attributes. Therefore, there exists a fairly deterministic relationship between the features and the target, which can be obtained by a highly customized preprocessing step. However, the performance improvement is not the key objective of our experiments, so minimal preprocessing is performed.

The topmost row in Figure 8 presents the results from the Amazon Employee Dataset experiment. Unlike the previous datasets (simulated, Mercedes-Benz, and Crime), the AUROC of PaloBoost, SGTB-Bonsai, and XGBoost are still improving after 200 iterations. Only Scikit-Learn demonstrate overfitting on the dataset. We suspect there are two reasons for this behavior. Since all the features in this dataset are categorical, the degree of freedom in the input space is finite, thereby reducing the risk of the curse of dimensionality. Secondly, the fairly deterministic nature of the features and the targets combined with the large sample size make it less prone to overfitting.

Another important observation from Figure 8 is that PaloBoost converges slower compared to the other models on this dataset. This is because PaloBoost has two additional regularization mechanisms: gradient-aware pruning and adaptive learning rates. While these modifications are likely to improve the performance for a complex problem (e.g., large number of features, small dataset size, large amount of noise, etc.), it can hinder the performance on simpler problems. By regularizing the trees and the learning rates, PaloBoost is unable to quickly learn an appropriate model and thereby requires more iterations to achieve the same accuracy[7]. Therefore, PaloBoost may not be appropriate when standard SGTB models are not prone to overfitting.

4.6.2 Pulsar Detection Dataset. Pulsars, rapidly rotating neutron stars, are known to emit a detectable pattern of electromagnetic radiation. Unfortunately, automated detection based on the pattern is often a non-trivial task due to radio frequency interference and noise. Nowadays, machine learning techniques are adopted to

[6]https://www.kaggle.com/c/amazon-employee-access-challenge
[7]PaloBoost eventually achieves comparable AUROC.



(a) Amazon, $\nu_{max} = 1.0$ (b) Amazon, $\nu_{max} = 0.5$ (c) Amazon, $\nu_{max} = 0.1$

(d) Pulsar, $\nu_{max} = 1.0$ (e) Pulsar, $\nu_{max} = 0.5$ (f) Pulsar $\nu_{max} = 0.1$

(g) Carvana, $\nu_{max} = 1.0$ (h) Carvana, $\nu_{max} = 0.5$ (i) Carvana, $\nu_{max} = 0.1$

(j) BNP, $\nu_{max} = 1.0$ (k) BNP, $\nu_{max} = 0.5$ (l) BNP, $\nu_{max} = 0.1$

Fig. 8. Predictive performance, measured in AUROC, for the four classification tasks. From the top, at each row, we have Amazon Employee Access Dataset (1st row), Pulsar Detection Dataset (2nd row), Carvana Dataset (3rd row), and BNP-Paribas Cardif Dataset (4th row). The y-axis is aligned for each dataset (row) for ease of comparison across learning rates.

extract candidates for detecting pulsars. In the effort to develop such machine learning detection algorithms, Lyon curated and published the HTRU2 (High Time Resolution Universe Survey - South) dataset [30, 31]. The dataset, published on UCI[8], contains the profiles of electromagnetic waves and the label (pulsar or not) of 17,898 examples, where 1,639 examples are pulsars (positive class). All eight features are numeric and have no missing values (see Table 1).

The second row in Figure 8 illustrates the results from the Pulsar Detection Dataset experiment. As can be seen, PaloBoost exhibits the most stable and best performance overall. However, the performance differences are practically indistinguishable for smaller learning rates. This may be a result of the dataset curation process, where only engineered features that are strongly correlated with the target variable were included. Even a single, albeit highly customized, decision tree algorithm could achieve a reasonable performance [31]. In other words, the dataset has no noisy features, thus overfitting is less likely to happen even for complex models.

4.6.3 Carvana Dataset. When auto dealers purchase used cars from an auction, there is a risk of purchasing cars with serious issues such as tampered odometers and mechanical issues, which is commonly referred as "kicks" in the industry. Carvana is an online used-car dealers company. The company released its dataset on Kaggle[9] that contains used cars' attributes, such as make, model, vehicle age, odometer, etc., and the target variable (kick or not). The dataset (summarized in Table 1) has 72,983 samples, with 151 numeric features (after preprocessing). Due to missing values, Scikit-Learn is omitted from this experimental study.

The third row in Figure 8 shows the results from the dataset. As can be seen, the two baseline models clearly display overfitting behaviors for all three learning rates. Moreover, the dataset is prone to overfitting, as illustrated by the Kaggle leaderboards. A comparison of the public and private leaderboards[10] shows considerable movement between the top 4 teams. This indicates the top teams' models likely suffered from overfitting. Notably, PaloBoost exhibits extremely stable predictive performances and records the best scores for all cases. Moreover, the difference in performance across the three learning rates is almost negligible. Although not shown, the prune and adaptive learning rates displayed similar patterns to the simulated dataset (Figure 3).

4.6.4 BNP Paribas Cardif Claims Dataset. Insurance claim payments involve many levels of manual checks, sometimes requires more information to be collected, and can take a substantial amount of time till the final payments. Some of these claims, however, can be processed much faster with machine learning techniques. BNP Paribas Cardif, one of the largest personal insurance company, released some of its data via the Kaggle platform[11] that contains various claim attributes (features) and if the claim's approval could be accelerated or not (target). The dataset consists of 114,321 samples with 273 numeric features (after preprocessing). Scikit-Learn is omitted from this study as the dataset has many missing values (see Table 1). This dataset contains the largest number of samples (of the seven datasets) and shows a fair degree of complexity.

The last row in Figure 8 illustrates the results from the BNP Paribas Cardif Claims Dataset experiment. Similar to all but the Amazon classification task, PaloBoost offers the most stable and best predictive performance. The competitors involved in this Kaggle challenge remarked that they suspect the dataset has additional noise to the features[12]. Many winning solutions were based on extensive feature engineerings, identifying the relationships of the features, and removing the noisy features. The impressive performance from PaloBoost suggests the ability

[8]https://archive.ics.uci.edu/ml/datasets/HTRU2
[9]https://www.kaggle.com/c/DontGetKicked
[10]Kaggle sets one-third of test data for ranking the public leaderboard, and the rest for the private leaderboard. The private leaderboard is closed during the competition and announced after the competition is closed. This is to prevent the overfitting on the test set [22].
[11]https://www.kaggle.com/c/bnp-paribas-cardif-claims-management
[12]See discussions 19240 and 20247 on the Kaggle discussion board at https://www.kaggle.com/c/bnp-paribas-cardif-claims-management/discussion/ as some examples.

to effectively identify the noisy features to yield a more robust model. This also suggests that PaloBoost can outperform other SGTB implementations when a minimal preprocessing is applied to a noisy dataset.

Based on our extensive experiments, PaloBoost outperforms existing SGTB models when a dataset has many noisy features and is prone to overfitting. While PaloBoost is not guaranteed to achieve the best predictive performance on all datasets (see the Amazon results in the top row of Figure 8), it is less prone to overfitting. From Figures 2, 7, and 8), we observe that PaloBoost exhibits a gradual performance degradation in all seven datasets, unlike the other baseline models. Moreover, the predictive performance of PaloBoost using different learning rates is relatively small. Thus, we conclude that PaloBoost is robust to the specification of the learning rate (ν_{max}), the tree depth, and also the number of iterations M.

5 DISCUSSIONS

We introduced PaloBoost, an extension of SGTB, to mitigate overfitting and minimize exhaustive hyperparameter tuning. PaloBoost uses two regularization techniques to perform 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 an alternative training sample. Based on this new perspective of the under-utilized OOB samples, PaloBoost can dynamically adjust the tree depths and learning rates at each stage. With these two mechanisms, PaloBoost can automatically adapt to minimize overfitting by knowing when it needs to "slow down". Furthermore, by introducing efficient computations, these regularizations can be readily implemented on top of SGTB with minimal computational impact.

Our extensive experiments using both real and simulated datasets confirm that PaloBoost produces robust predictive performance. In particular, when a dataset is noisy and complex, PaloBoost significantly outperforms the other SGTB implementations. We also show that our two regularization mechanisms can adaptively adjust to guard against overfitting by yielding lower variance trees, and optimal region-specific learning rates. Moreover, the empirical results demonstrate considerably less sensitivity to the parameter settings – different learning rates and the number of iterations yield comparable predictive performance.

We also introduced a new feature importance formula that showed promising results on the Friedman's Simulated Dataset. While other SGTB implementations failed to identify the relevant features, PaloBoost's importance estimates accurately captured the true importances. We posit that the OOB regularizations (removing noisy nodes) in conjunction with accounting for the coverage of the region and the node estimates, yields a superior feature selection process. In addition, we proposed a new visualization based on feature importance to identify the optimal number of iterations. It would be interesting to see if the feature selection mechanisms from PaloBoost can help other machine learning algorithms.

Another observation drawn from the experiments was the presence of many trees with negligible (close to zero) 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.

PaloBoost is developed to make the SGTB more robust and stable. While significantly less sensitive to the hyperparameters compared to the other implementations, PaloBoost still requires similar parameters ("max" learning rate and tree depth). We note that the ideas presented in this paper are the initial steps towards 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] Salford Systems: a Minitab company. 2018. TreeNet - Gradient Boosting. https://www.salford-systems.com/products/treenet.

[2] Amazon.com. 2013. Kaggle Competition: Amazon.com - Employee Access Challenge. https://www.kaggle.com/c/amazon-employee-access-challenge.
[3] Elnaz Barshan and Paul Fieguth. 2015. Stage-wise Training: An Improved Feature Learning Strategy for Deep Models. In *Proceedings of the 1st International Workshop on Feature Extraction: Modern Questions and Challenges at NIPS 2015 (Proceedings of Machine Learning Research)*, Dmitry Storcheus, Afshin Rostamizadeh, and Sanjiv Kumar (Eds.), Vol. 44. PMLR, Montreal, Canada, 49–59. http://proceedings.mlr.press/v44/Barshan2015.html
[4] Ron Bekkerman. 2015. The Present and the Future of the KDD Cup Competition. https://www.kdnuggets.com/2015/08/kdd-cup-present-future.html.
[5] Leo Breiman. 1996. Bagging Predictors. *Machine Learning* 24, 2 (1996), 123–140.
[6] Leo Breiman. 1998. Arcing classifier (with discussion and a rejoinder by the author). *The Annals of Statistics* 26, 3 (June 1998), 801–849.
[7] Leo Breiman. 2001. Random Forests. *Machine Learning* 45, 1 (2001), 5–32.
[8] Chris J.C. Burges. 2010. *From RankNet to LambdaRank to LambdaMART: An Overview*. Technical Report. https://www.microsoft.com/en-us/research/publication/from-ranknet-to-lambdarank-to-lambdamart-an-overview/
[9] BNP Paribas Cardif. 2015. Kaggle Competition: BNP Paribas Cardif Claims Management. https://www.kaggle.com/c/bnp-paribas-cardif-claims-management.
[10] Carvana. 2012. Kaggle Competition: Don't Get Kicked! https://www.kaggle.com/c/DontGetKicked.
[11] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 785–794.
[12] Dua Dheeru and Efi Karra Taniskidou. 2017. UCI Machine Learning Repository. http://archive.ics.uci.edu/ml
[13] Anna Veronika Dorogush, Vasily Ershov, and Andrey Gulin. 2017. CatBoost: gradient boosting with categorical features support. *Workshop on ML Systems at NIPS 2017* (2017). arXiv:82723822-0E73-4D5C-A16D-36A3A4A30A89
[14] Bradley Efron and Gail Gong. 1983. A Leisurely Look at the Bootstrap, the Jackknife, and Cross-Validation. *The American Statistician* 37, 1 (February 1983), 36–48.
[15] Jane Elith, Jone R. Leathwick, and Trevor Hastie. 2008. A working guide to boosted regression trees. *Journal of Animal Ecology* 77, 4 (July 2008), 802–813.
[16] Yoav Freund and Robert E. Schapire. 1999. A Short Introduction to Boosting. *Journal of Japanese Society for Artificial Intelligence* 14, 5 (September 1999), 771–780.
[17] Jerome H. Friedman. 1991. Multivariate Adaptive Regression Splines. *The Annals of Statistics* 19, 1 (March 1991), 1–67.
[18] Jerome H Friedman. 2001. Greedy function approximation: A gradient boosting machine. *The Annals of Statistics* 29, 5 (Oct. 2001), 1189–1232.
[19] Jerome H Friedman. 2002. Stochastic gradient boosting. *Computational Statistics & Data Analysis* 38, 4 (Feb. 2002), 367–378.
[20] Jerome H. Friedman. 2006. Recent Advances in Predictive (Machine) Learning. *Journal of Classification* 23, 2 (September 2006), 175–197.
[21] Jerome H. Friedman, Eric Grosse, and Werner Stuetzle. 1983. Multidimensional Additive Spline Approximation. *SIAM J. Sci. Statist. Comput.* 4, 2 (1983), 291–301.
[22] Moritz Hardt and Avrim Blum. 2015. The ladder: a reliable leaderboard for machine learning competitions. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning*, Vol. 37. 1006–1014.
[23] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. 2009. *The elements of statistical learning: data mining, inference, and prediction* (2 ed.). Springer.
[24] Xinran He, Junfeng Pan, Ou Jin, Tianbing Xu, Bo Liu, Tao Xu, Yanxin Shi, Anoine Atallah, Ralf Herbrich, Stuart Bowers, and Joaquin Quinonero Candela. 2014. Practical Lessons from Predicting Clicks on Ads at Facebook. In *Proceedings of the Eighth International Workshop on Data Mining for Online Advertising*. 1–9.
[25] Gordon F. Hughes. 1968. On the mean accuracy of statistical pattern recognizers. *IEEE Transactions on Information Theory* 14, 1 (January 1968), 55–63.
[26] Kaggle. 2017. Kaggle.com. https://www.kaggle.com.
[27] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. LightGBM: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems 30*. 3146–3154.
[28] Ron Kohavi. 1995. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings of the 14th international joint conference on Artificial intelligence*, Vol. 2. 1137–1143.
[29] Ping Li, Chris J.C. Burges, and Qiang Wu. 2008. Learning to Rank Using Classification and Gradient Boosting. In *Advances in Neural Information Processing Systems*. MIT Press, Cambridge, MA. https://www.microsoft.com/en-us/research/publication/learning-to-rank-using-classification-and-gradient-boosting/
[30] Rrobert J. Lyon. 2016. HTRU dataset: High Time Resolution Universe Survey (South). https://doi.org/10.6084/m9.figshare.3080389.v1
[31] Robert J. Lyon, B. W. Stappers, S. Cooper, J. M. Booker, and J. D. Knowles. 2016. Fifty Years of Pulsar Candidate Selection: From simple filters to a new principled real-time classification approach. *Monthly Notices of the Royal Astronomical Society* 459, 1 (2016), 1104–1123. https://doi.org/10.1093/mnras/stw656

[32] Mercedes-Benz. 2017. Kaggle Competition: Mercedes-Benz Greener Manufacturing. https://www.kaggle.com/c/mercedes-benz-greener-manufacturing.
[33] Yubin Park. [n. d.]. Bonsai-dt - programmable decision tree framework. https://yubin-park.github.io/bonsai-dt/.
[34] Fabian Pedregosa, Gael Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Edouard Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.
[35] Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. 2017. CatBoost: unbiased boosting with categorical features. *arXiv:1706.09516 [cs.LG]* (June 2017). arXiv:1706.09516v2
[36] Michael A. Redmond and Alok Baveja. 2002. A Data-Driven Software Tool for Enabling Cooperative Information Sharing Among Police Departments. *European Journal of Operational Research* 141 (2002), 660–678.
[37] Greg Ridgeway. 2012. Generalized boosted models: A guide to the gbm package. (2012).
[38] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. 2015. FitNets: Hints for Thin Deep Nets. https://arxiv.org/abs/1412.6550.
[39] Holger Schwenk and Yoshua Bengio. 2000. Boosting Neural Networks. *Neural Computation* 12, 8 (August 2000), 1869–1887.
[40] Kai Ming Ting and Lian Zhu. 2009. Boosting Support Vector Machines Successfully. In *International Workshop on Multiple Classifier Systems*. 509–518.
[41] Gerard V. Trunk. 1979. A Problem of Dimensionality: A Simple Example. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 1, 3 (July 1979), 306–307.
[42] He Zhang and Vishal M. Patel. 2018. Densely Connected Pyramid Dehazing Network. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 3194–3203.
[43] Hui Zou and Trevor Hastie. 2005. Regularization and Variable Selection via the Elastic Net. *Journal of the Royal Statistical Society, Series B (Statistical Methodology)* 67, 2 (2005), 301–320.