Gradient Boosting Regression Trees with Variable Shrinkage

Austin Barket
Department of Computer Science
The Pennsylvania State University at Harrisburg
Middletown, PA 17057
Email: amb6470@psu.edu

Jeremy Blum
Department of Computer Science
The Pennsylvania State University at Harrisburg
Middletown, PA 17057
Email: jjb24@psu.edu

***Abstract*—In this research we explore the effect of variable shrinkage (learning rate) on gradient boosting machines that utilize regression trees as the base learners. Conventionally, research and implementations of gradient boosting machines have used only constant shrinkage that is applied uniformly to the predictions of all base learners.**

I. Introduction

Boosting is a machine learning technique that combines learning algorithms that barely beat random guessing, known as a weak or base learners, into a single model with significantly improved accuracy or lower error rates over any of its constituent parts [1] [2].

The gradient boosting machine (GBM), originally introduced by Friedman in 1999 is a general boosting framework that leverages the steepest descent numerical optimization method to iteratively train base learners to address the errors made by those before them [3].

Production implementations of gradient boosting machines such as the gbm package in R have found remarkable traction among researchers in a wide variety of fields including robotics and ecology [4] [5]. Interestingly these packages generally implement Friedman's Gradient Boost algorithm as it was originally defined [6], leaving some definite opportunity for research into algorithmic tweaks to improve performance.

A particular component of the algorithm that has not been explored to date is the learning rate, commonly referred to as shrinkage in the case of GBMs. Shrinkage is traditionally implemented as a constant parameter to the model. After each iteration the new base learner's predictions are scaled by the shrinkage as a form of regularization.

This proposal outlines a new way to think about shrinkage for the common case where the base learners are regression trees. We hypothesize that by varying the shrinkage applied to the prediction of the examples in each individual leaf of the regression tree base learners, we will be able to decrease convergence time without sacrificing resilience to overfitting.

II. Related Work

Boosting finds its roots in a question originally posed by Kearns and Valient in 1988, is weak learnability equivalent to strong learnability [7] [8]? In other words, if you have a way to learn well enough to beat random guessing, is it inherently true that a strong learner, capable of arbitrarily low error, for that same problem exists? Schapire successfully proved this equivalence in 1990 by proposing and proving the correctness of a polynomial time boosting model he termed *The hypothesis boosting mechanism* [1].

After Schapire's compelling proof that weak and strong learnability are in fact equivalent, researchers bagan working to improve upon his boosting algorithm. The first practical application of the early boosting algorithms came out of the work of Drucker, Schapire, and Simard at AT&T Bell Labs in 1992. There they applied boosting of neural network base learners to the problem of optical character recognition of handwritten zip codes on USPS letters [9].

In 1995 Freund and Schapire introduced the AdaBoost algorithm which is hailed as solving many of the practical problems suffered by previous boosting algorithms. The unique idea introduced by Adaboost is the notion of applying higher weights at each iteration to the training examples that were misclassified in previous iterations, forcing the new base learners to focus their efforts on these examples. AdaBoost became famous as an excellent out of the bag approach for classification with exceptional resilience to overfitting [10]. However, the details of why exactly AdaBoost worked were unknown until the year 1998 when Friedman, Hastie, and Tibshirani explored the algorithm from an in depth statistical viewpoint. They found that AdaBoost a specialized additive model, and applied knowledge from the long history of statistical additive function approximation to gain a better understanding of AdaBoost and boosting in general [11].

With an increased theoretical statistical understanding of boosting now available, Friedman developed a generalized stagewise additive function approximation boosting model termed the gradient boosting machine in 1999, which he later extended to include a stochastic subsampling approach in 2002 [3] [12]. Gradient Boosting Machines will be explored is great detail in the following sections as extending this model is the focus of the proposed research.

Since their introduction to the machine learning and data mining communities in 1999, gradient boosting machines have found applications in a variety of fields for both classification and regression tasks. Most recently ecology researchers have found great interest in gradient boosting machines, particularly the varient of them that utilizes classification or regression trees as base learners. In 2007, Glenn De'ath extended the R package gbm, creating a new package gbmplus that implements a varient algorithm he terms Aggregated Boosted Trees (ABT). The idea behind ABTs is to perform cross validation to determine an optimal number of iterations for the boosting, then save the models built during cross validation chopping them off at the optimal number of iterations found. To make a prediction, the predictions of all of these boosted trees are computed then averaged. It was found that this approach lead to improved accuracy over gbm alone [13].

Another group of ecological researchers Jane Elith and John Leathwick have also been applying boosted regression trees to their work. One such problem involves predicting whether or not a particular species off eel will be present in unsampled Australian rivers based upon measured environmental factors [5]. Elith and Leathwick implemented their own extensions to the functions in the gbm package in their dismo package in 2015, including a very useful function known as gbm.step. This function implements a cross validation based approach to estimate the number of trees that should be built to reach optimal generalization error, an approach that we will mimic in our experiment [14].

III. The Gradient Boosting Machine

In supervised learning, our goal is to find an approximation $\hat{F}$ of an unknown function $F : \vec{x} \rightarrow y$ that maps data instances $\vec{x}$ to a set of response variables y and best minimizes the expected value of some loss function $\Psi(y, F(\vec{x}))$. Friedman's gradient boosting machine achieves this by iteratively constructing a strong learner that approximates F from many weak learners also referred to as base learners. $h(\vec{x})$ is the standard notation for the generic base learner, which takes an instance $\vec{x}$ and predicts the value of its response variable. In each iteration a new base learner, such as a short regression tree, is trained to fit the errors made by the function approximation so far. This training is based upon an extremely common numerical minimization method known as steepest gradient descent [6] [3] . However, unlike most applications of steepest descent, Friedman's Gradient Boost algorithm computes the negative gradient $\vec{g}$ in the space of the estimated function itself, not in the space of a finite set of parameters that define the function. By framing the problem in this way, the function $\hat{F}$ is not limited to a set of functions definable by a finite set of parameters, but rather is defined by a potentially infinite set of parameters, one for each possible value $\vec{x}$. Obviously, it is impossible to actually compute the gradient and apply steepest-descent in this potentially infinite dimensional function space, but it is possible to perform steepest-descent with respect to the finite space of training examples D [6] [3].

The negative gradient in this restricted subset of function space defines the direction of steepest descent in the loss function for the training examples. Thus by updating the function $\hat{F}$ directly by this negative gradient, we would move closer to the minimum values of the loss function Ψ for the examples in the training dataset. Of course this is not quite the goal, instead we would like to be able to generalize to all possible data. To accomplish this we instead train a base learner to predict the negative gradient of the loss function at each step, then update our function with this model's prediction. Friedman's general Gradient Boosting Machine, extended to include his later ideas of subsampling the training data and applying a constant shrinkage to improve generalization is provided in Algorithm 1 [6] [3] [12].

```
input : Training Dataset: $D = (x_i, y_i),\ i = 1...N$
        Bag Fraction: $bf \,\epsilon\, [0, 1]$
        Shrinkage: $v \,\epsilon\, [0, 1]$
        Number of Base Learners: $M$
        Loss Function: $\Psi$ e.g. RMSE
        Choice of Base Learner: $h(\vec{x})$ e.g. regression trees
output: A function $\hat{F}(\vec{x})$ that minimizes the expected value of
        $\Psi(y, F(\vec{x}))$

Initialize the approximation of $\hat{F}$
    $\hat{F}_0(\vec{x}) = argmin_\rho \sum_{i=1}^{N} \Psi(y_i, \rho)$    (1)
for $m \leftarrow 1$ to $M$ do
    Select a random subsample, $S_m$, of size $\tilde{N}$ from the
    training data without replacement.
        $S_m \subset D, \quad |S_m| = \tilde{N} = bf \cdot N$    (2)
    Approximate the negative gradient $\vec{g}_m$ of $\Psi(y_i, \hat{F}_{m-1}(\vec{x}))$
    with respect to $\hat{F}_{m-1}(\vec{x})$.
        $g_{m,i} = -\frac{\partial}{\partial \hat{F}_{m-1}(\vec{x_i})} \Psi(y_i, \hat{F}_{m-1}(\vec{x_i})),\ \vec{x}_i \,\epsilon\, S_m$    (3)
    Train a new base learner $h_m(\vec{x})$ to predict $\vec{g}_m$ and fit the
    least squares.
        $\beta_m, h_m(\vec{x}) = argmin_{\beta, h(\vec{x})} \sum_{\vec{x}_i \,\epsilon\, S_m} [g_{m,i} - \beta h(\vec{x}_i)]^2$    (4)
    Solve for the optimal coefficient $\rho$ that minimizes $\Psi$.
        $p_m = argmin_\rho \sum_{\vec{x}_i \,\epsilon\, S_m} \Psi(y_i, \hat{F}_{m-1}(\vec{x}_i) + \rho h(\vec{x}_i))$    (5)
    Update your approximation of $\hat{F}$, scaled by the shrinkage v
        $\hat{F}_m(\vec{x}) = \hat{F}_{m-1}(\vec{x}) + v \cdot \rho_m h_m(\vec{x})$    (6)
end
```

Algorithm 1: Friedman's Gradient Boost Algorithm [3] [4] [12] [13]

IV. Simplifying Assumptions

For the purposes of the current research, we will consider only the case where Ψ is the squared error function and the goal is to predict a real valued response variable. Without loss of generality we will add a coefficient of $\frac{1}{2}$ to Ψ so that the negative of the partial derivative of Ψ with respect to the predicted value of an instance $\vec{x}_i$ is simply the residual of that prediction. In this case the component wise calculation of the negative gradient (Equation 3) becomes Equation 7.

$$g_{m,i} = -\frac{\partial}{\partial \hat{F}_{m-1}(\vec{x_i})} \frac{(y_i - \hat{F}_{m-1}(\vec{x_i}))^2}{2} = (y_i - \hat{F}_{m-1}(\vec{x_i})) \quad (7)$$

As mentioned in the introduction, the proposed variable shrinkage scheme is specially designed for the common case where the base learners are regression trees, we define the notation for regression trees in Figure 1 below.

When building the trees, we define the next best split as the split that leads to the largest reduction in squared error on the training examples. Thus the least squares coefficient β in Equation 4 will always be 1, and since we're using the squared error loss function so will the optimal coefficient ρ in Equations 5 and 6.

Thus, for the case where the base learners are regression trees and the loss function is the squared error, the update step in Equation 6 can be replaced by Equation 8

$$\hat{F}_m(\vec{x}) = \hat{F}_{m-1}(\vec{x}) + v \cdot \sum_{j=1}^{J} b_{m,j} I(\vec{x} \,\epsilon\, R_{m,j}) \quad (8)$$

The choice of regression tree base learners introduces two additional parameters to the gradient boosting algorithm, namely the maximum number of splits in each tree (aka interaction depth) and the minimum number of observations in each leaf node (leaf size). These parameters work together to regularize the complexity of the regression trees, and in general these parameters are chosen to ensure short trees and stout leaves to weaken the predictive power of each individual base learner which ultimately strengthens the boosted model's ability to generalize [5].

$$h_m(\vec{x}) = \sum_{j=1}^{J} b_{m,j} I(\vec{x} \,\epsilon\, R_{m,j}) \tag{9}$$

Where

$J =$ the number of terminal nodes (leaves) in the tree

$b_{m,j} =$ Prediction made for all instances in $R_{m,j}$.
For squared error, $b_{m,j} = avg_{x_i \epsilon R_{m,j}}(g_{m,i})$

$R_{m,j} =$ The subset of instances $\vec{x} \,\epsilon\, S_m$
that are predicted by the j^{th} terminal node.

$$I(\alpha) = \begin{cases} 1 & \alpha \text{ is true} \\ 0 & \alpha \text{ is false} \end{cases}$$

Fig. 1: Notation for Regression Tree Base Learners

Note that although we restrict the current study to regression tasks using the squared error loss function, the variable shrinkage scheme defined and tested in the following sections could easily be applied to any of the loss functions originally defined by Friedman in [3] [12], and most famously implemented in the gbm R package originally written by Ridgeway [6]. Please see these references for information on the many specialized derivations of Algorithm 1 for various learning tasks and to better understand the reasoning behind the equations defined in this section.

V. Variable Shrinkage for Regression Tree Base Learners

The use of regression trees presents an interesting possibility of a simple, yet elegant adaptation method. Since the trees themselves can be seen as a summation of individual prediction terms, one for each leaf, a natural adaptation scheme is to compute a different shrinkage for each leaf node in each base learner. Specifically a simple linear mapping can be used to ensure that the lower the number of examples in a given leaf node, the lower the computed shrinkage. We hypothesize that this will encourage the training to learn rapidly from the most typical training examples, which will fall into large leaf nodes along with their similar peers, while diminishing the impact of outlying and noisy examples, which will be isolated by the regression tree's splitting algorithm. Overall we expect this scheme to lead to faster convergence time while strongly discouraging overfitting.

To this end we will alter Algorithm 1 to take as input both a minimum and maximum shrinkage v_{min} and v_{max}, instead of the constant shrinkage v. The following equation will then be used to compute the shrinkage for each of the J leaves in Equation 8.

$$v_{m,j} = \frac{|R_{m,j}|}{|S_m|}(v_{max} - v_{min}) + v_{min} \tag{10}$$

This equation maps the range of possible leaf node sizes $[\ 0, \tilde{N}\ ]$ to the range of possible shrinkage values $[\ v_{min}, v_{max}\ ]$. Note that even when the regression trees are built with a nonzero minimum number of observations in each leaf node, the missing value branches of the trees can still have any number of examples in them, including zero if no missing values existed in the training data. Thus zero is still used as the minimum leaf size for the purposes of shrinkage calculation regardless of the minimum number of observations parameter.

Using equation 10, the update step (Equation 8) becomes Equation 11.

$$\hat{F}_m(\vec{x}) = \hat{F}_{m-1}(\vec{x}) + \rho_m \sum_{j=1}^{J} v_{m,j} \cdot b_{m,j} I(\vec{x} \,\epsilon\, R_{m,j}) \tag{11}$$

VI. Implementation

We implemented the Friedman's gradient boosting machine with the option to utilize either the standard constant shrinkage or our proposed variable shrinkage scheme as defined in Sections III, IV, and V. The core algorithm implementation is an extension of the very minimal JSGBM project found here [15]. The additions to the original project include support for missing values and splits on categorical variables, optimization of the splitting algorithm, calculation of predictor relative influence, a cross validation based method for selecting the optimal number of trees, and support for De'ath's aggregated boosted trees as discussed in Section II. All code implemented for this paper can be found at [16].

The algorithm for selecting the optimal number of trees is based on the gbm.step function of the dismo R package which is described in [5]. The psuedocode for our modified implementation in provided in Algorithm 2.

```
input : Number of Folds: $K = 5$
        Step Size: $S_{size} = 500$:
        Steps Past Minimum: $S_{pm} = 3$
        Max Number of Trees: $M = 150,000$

output: Estimated number of trees at which generalization error
        is minimized

Create $K$ training/validation set pairs as in normal K-fold cross
validation. For each pair, create a new empty GBM model.

$S_{remaining} \leftarrow S_{pm}$ (Initialize number of steps remaining)

while NumberOfTrees $< M$ and $S_{remaining} > 0$ do
    Add $S_{size}$ trees to each of the $K$ GBM models.
    $avgRMSE \leftarrow$ Evaluate the new validation RMSE of each
    model, and average them.
    if $avgRMSE < minAvgRMSE$ then
        $S_{remaining} \leftarrow S_{pm}$
        $minAvgRMSE \leftarrow avgRMSE$
    else
        $S_{remaining} \leftarrow S_{remaining} - 1$
end
```

Algorithm 2: Find Optimal Number of Trees using Cross Validation

VII. Datasets

Four real world datasets with natural regression tasks were selected for use in this study. Each of these can be found in the UC Irvine Machine Learning Repository [17]. This section introduces each of these in turn. Note the parenthesized minimal name found in the subsection titles will be used to refer to these datasets throughout the results and discussion sections.

A. Combined Cycle Power Plant (powerPlant)

This dataset contains 9568 examples collected over 6 years from a power plant set work at full load. Each instance consists of 4 real valued predictors; namely the Temperature (AT), Ambient Pressure

(AP), Relative Humidity (RH) and Exhaust Vacuum (V). The target response variable is the net hourly electrical energy output (EP) of the plant [18]. The powerPlant dataset was subjected to extensive cleaning and preprocessing prior to donation to the UCI repository. This preprocessing procedure filtered out all nonsensical outliers and diminish the noise that resulted from electrical disturbance interfering with the signal [19] [20].

B. *Airfoil Self-Noise (nasa)*

This data originated from a 1989 NASA experiment in which different size air foils (wing shapes) were placed in a wind tunnel and subjected to various free-stream velocities (wind speed prior to hitting the air foil) and angles of attack (direction of the wind). The regression goal is to predict the scaled sound pressure level in decibels of an air foil given the free-stream velocity (meters/second), angle of attack (degrees), frequency (Hertz), chord length (meters), and suction side displacement thickness (meters) [21]. More information on the creation and properties of this dataset can be found in the following relevant papers [22] [23] [24].

C. *Bike Sharing (bikeSharing)*

Hadi Fanaee from the Laboratory of Artificial Intelligence and Decision Support (LIAAD), University of Porto pulled together weather information and holiday information and integrated it with publically available trip data provided by Capital Bikeshare to generate this intriguing dataset consisting of almost two years of bike rental data aggregated both by day and by hour. Each instance contains attributes providing information about the day [and hour] that the rentals occurred. This includes the year, season, month, day, hour, whether it was a holiday, a weekday, or a working day. All of the time attributes are treated as categorical predictors in our experiment. In addition the weather conditions for each instant in time are provided including the precipitation type, temperature, humidity, and wind speed. Finally the count of bikes rented during each time period is provided, which will be used as the target variable in our experiment [25] [26].

For our purposes we will look only at the daily data which limits the dataset to 731 instances rather than 17381 instances in the hourly version. No normalization or data cleaning has been applied to this dataset. This dataset is of course naturally quite noisy since people do not follow any hard and fast rules when deciding whether or not to go on a bike ride so the rental counts can vary considerably even for identical values of all the predictors.

D. *Communities and Crime (crimeCommunities)*

Lastly we will look at the Communities and Crime dataset which contains 122 predictors for the ratio of violent crime to population in 1994 communities throughout the United States, where violent crimes are defined as murder, rape, robbery, and assault. Each example is comprised of data from the 1990 US Census, law enforcement data from the 1990 US LEMAS survey, and crime data from the 1995 FBI UCR. Some examples of the predictors provided are the household size, percent unemployed, divorce rates, ratio of police offices to population, number of different kinds of drugs seized by the police, average rent costs, percentage of variance races and ethnicities, and many more aspects of the community and local police force. Note that in this dataset all of the variables have been normalized to the range $[\ 0, 1\ ]$ and missing values are prevalent for many of the predictors. [27] [28] [29] [30] [31] [32].

Maximum Running Time	1.5 hours
Maximum Memory Usage	20 GB
Maximum Number of Trees	150,000
Number of CV Folds	5
CV Step Size	500
Max CV Steps w/o improving CV Error	3

TABLE I: Execution limits and Parameters for Finding Optimal Number of Trees

Bag Fraction	0.25, 0.50, 0.75, 1
Min Examples In Node	1, 10, 75, 150
Max Number Of Splits	1, 2, 4, 8, 16, 32, 64, 128
Constant: Shrinkage	0.1, 0.01, 0.001, 0.0001
Variable: Minimum Shrinkage	0.01, 0.001, 0.0001
Variable: Maximum Shrinkage	0.1, 0.4, 0.7, 1

TABLE II: Execution limits and Parameters for Finding Optimal Number of Trees

VIII. Experiment

In order to compare model performance of the original algorithm with our proposed shrinkage adaption scheme, we developed an experiment in which 2048 total sets of parameters were tested on the four datasets described in Section VII.

All parameters that affect the execution of Algorithm 2, which is the core training procedure used in all of our tests, are shown in Table I. Two additional parameters are listed that were not introduced with algorithm itself but are important to note. These indicate that the running time and memory usage for during training was limited to 1.5 hours and 20 GB respectively. These values were exceeded only a handful of times when using the lowest constant learning rate and the highest number of splits. In the rare case that a test exceeded these limits the training stopped and the application continued with collecting and saving metadata about the constructed models, just as though the training stopped due to the normal stopping condition.

Gradient boosting machines with regression tree base learners are traditionally defined by the number of trees to be built and 4 additional parameters; the bag fraction, shrinkage, minimum number of examples in each leaf, and the maximum number of splits in each tree. To support our variable shrinkage scheme we must add one additional parameter since we require both a minimum and a maximum learning rate. All values that were considered for each of these parameters are provided in Table II. Taking all possible combinations of these parameters leads to 512 parameter sets with constant shrinkage, and 1536 possibilities with variable shrinkage for a total of 2048 individual tests for each dataset. The exact procedure used for each of these tests is provided in Algorithm 3.

IX. Results

For each dataset, we have sorted all parameter sets by the average cross validation error across the five runs performed. The five sets of parameters leading to the lowest average cross validation root mean squared error (RMSE) using both constant and variable shrinkages are provided for each dataset in Tables III through X. Note the lighter shaded cells indicate the minimum value in each row, while the darker shade indicates the maximum value.

The topmost sections of these tables contain the parameter values themselves (columns are sorted by CV Error in ascending order).

In the middle portion of the table we have the running time, average cross validation error, average performance of the GBMs built with all training data on the test set (ATD Test RMSE), average performance

1) Select randomly without replacement 20% of the dataset and set aside as a test set.
2) Use Algorithm 2 to find the optimal number of trees. Simultaneously train a GBM using the entire training set, stop training when Algorithm 2's stopping condition is reached.
3) Build an Aggregated Boosted Tree from the K GBMs built for cross validation.
4) Collect and save meta-data about the run. This includes the optimal number of trees, the error values for each possible number of trees, the mean and standard deviation values for the actual number of splits and leaf sizes of each tree, the predictions made for each example at the optimal number of trees, and the average computed shrinkage values for each example and for each tree.
5) Compute the relative influences of the predictors for both the all training data GBM and the aggregated boosted tree.
6) Record the total amount of time taken to perform all of these steps.

Algorithm 3: Procedure for a Single Test.

Bag Fraction	0.5	0.5	0.5	0.5	0.75
Min Leaf Size	1.0	1.0	1.0	1.0	1.0
MaxSplits	32.0	32.0	16.0	64.0	16.0
Shrinkage	0.001	0.01	0.001	0.01	0.01
RunningTime (seconds)	335.1678	96.351	219.6743	81.041	90.7919
ATD Test RMSE	1.2844	1.2723	1.3036	1.2846	1.2429
ABT Test RMSE	1.3875	1.3934	1.3719	1.3639	1.3271
Cross Validation RMSE	1.4463	1.451	1.4542	1.4583	1.4686
Optimal Num. of Trees	137017.6	43745.8	149957.8	19801.6	59061.4
Avg. Leaf Size	6.37702	6.388851	13.570182	8.684288	18.810179
Std. Dev. Leaf Size	0.661731	2.80447	0.03642	7.786496	3.760119
Avg. Splits	29.6533	27.02	16	40.5662	14.7692
Std. Dev. Splits	3.0841	9.4346	0	26.7213	2.9538

TABLE III: Constant Shrinkage: Nasa Air Foil Parameters with Best Cross Validation RMSE.

of the Aggregated Boosted Trees on the test set (ABT Test RMSE), as well as the average optimal number of trees found across the five runs.

The bottom most sections of these tables contain the mean and standard deviation values of the actual leaf sizes and number of splits across all trees. In the case of variable shrinkage we additionally provide the average shrinkage applied across all of the examples.

It's important to notice that the trees cannot always be grown to the maximum number of splits or to the point of reaching the minimum leaf size. The tree building algorithm terminates when it can no longer find a split that decreases the overall squared error of the training instances involved in the split. Thus when the maximum number of splits is high, the minimum leaf size is low, th

X. Discussion

Analysis and stuff

XI. Conclusion

The conclusion goes here.

Bag Fraction	0.5	0.5	0.5	0.5	0.5
Min Leaf Size	1.0	1.0	1.0	1.0	1.0
MaxSplits	16.0	16.0	32.0	32.0	16.0
Min Shrinkage	0.01	0.001	0.0001	0.001	0.001
Max Shrinkage	0.7	0.4	0.4	0.4	0.1
RunningTime (seconds)	84.3753	198.8264	231.5566	168.5468	205.0479
ATD Test RMSE	1.2732	1.2448	1.1749	1.4282	1.3342
ABT Test RMSE	1.3357	1.2664	1.2811	1.5203	1.4044
Cross Validation RMSE	1.4229	1.4349	1.4353	1.4358	1.4465
Optimal Num. of Trees	56181	142271.4	98964.2	67466	149497.8
Avg. Leaf Size	12.750963	12.946516	5.347274	5.733489	13.572058
Std. Dev. Leaf Size	2.23063	1.670755	1.720047	1.538506	0.034655
Avg. Splits	15.0154	15.2533	24.8615	26.6667	16
Std. Dev. Splits	2.6282	1.9697	8.0028	7.1554	0
Avg. Shrinkage	0.177175	0.105513	0.067341	0.065603	0.028777
Std. Dev. Shrinkage	0.030301	0.020138	0.017297	0.016133	0.005734

TABLE IV: Variable Shrinkage: Nasa Air Foil Parameters with Best Cross Validation RMSE.

Bag Fraction	0.25	0.25	0.25	0.75	0.5
Min Leaf Size	1.0	1.0	1.0	1.0	1.0
MaxSplits	16.0	32.0	64.0	8.0	8.0
Shrinkage	0.001	0.001	0.001	0.0001	0.0001
RunningTime (seconds)	56.504	90.2223	235.1741	218.2886	192.6188
ATD Test RMSE	651.4022	658.0197	656.9807	676.6008	622.1279
ABT Test RMSE	655.1055	660.1975	661.3023	685.5282	623.8224
Cross Validation RMSE	621.5137	623.1882	623.9958	625.4049	626.079
Optimal Num. of Trees	8096	8025.4	26047.4	129637.2	121190
Avg. Leaf Size	3.302786	1.542667	0.329561	17.878478	11.507763
Std. Dev. Leaf Size	0.001619	0.268327	0.280136	2.725598	3.101209
Avg. Splits	16	29.4397	24.9575	7.4133	7.1467
Std. Dev. Splits	0.0025	5.1199	21.2159	1.1301	1.9259

TABLE V: Constant Shrinkage: Bike Sharing By Day Parameters with Best Cross Validation RMSE.

Bag Fraction	0.5	0.5	0.25	0.25	0.25
Min Leaf Size	1.0	1.0	1.0	1.0	1.0
MaxSplits	8.0	8.0	128.0	16.0	128.0
Min Shrinkage	0.01	0.001	0.0001	0.01	0.001
Max Shrinkage	0.1	0.1	0.4	0.1	0.4
RunningTime (seconds)	29.6724	28.487	158.0349	45.789	142.5841
ATD Test RMSE	650.0301	658.3118	653.8457	623.0296	670.386
ABT Test RMSE	651.2336	657.3432	645.1478	603.3586	671.631
Cross Validation RMSE	617.6814	618.1792	623.571	628.0958	628.7687
Optimal Num. of Trees	778	1976.4	4391.6	593	1859.6
Avg. Leaf Size	12.881251	12.881854	0.452567	3.302758	0.511332
Std. Dev. Leaf Size	0.019336	0.044956	0.130897	0.005692	0.012704
Avg. Splits	7.9999	7.9999	93.5407	15.9998	106.1888
Std. Dev. Splits	0.0071	0.0122	26.4443	0.0164	1.8836
Avg. Shrinkage	0.035531	0.031621	0.004295	0.019985	0.004885
Std. Dev. Shrinkage	0.001633	0.002721	0.000183	0.00108	0

TABLE VI: Variable Shrinkage: Bike Sharing By Day Parameters with Best Cross Validation RMSE.

Bag Fraction	0.75	0.75	0.75	0.5	0.75
Min Leaf Size	1.0	1.0	1.0	1.0	1.0
MaxSplits	16.0	64.0	32.0	32.0	64.0
Shrinkage	0.001	0.01	0.01	0.001	0.001
RunningTime (seconds)	1535.213	520.0124	374.4487	1509.8043	1713.554
ATD Test RMSE	2.9434	2.9665	2.8883	3.0721	2.9603
ABT Test RMSE	2.9368	2.9644	2.8819	3.0609	2.9751
Cross Validation RMSE	3.0016	3.0086	3.0226	3.0227	3.0307
Optimal Num. of Trees	149834.6	5613.8	9749.6	90980.4	54872.4
Avg. Leaf Size	130.42424	33.36434	66.21538	37.997457	30.981173
Std. Dev. Leaf Size	0	0	0	12.051963	6.956236
Avg. Splits	16	64	32	27.5478	59.4286
Std. Dev. Splits	0	0	0	8.7376	13.3436

TABLE VII: Constant Shrinkage: Power Plant Parameters with Best Cross Validation RMSE.

References

[1] R. E. Schapire, "The strength of weak learnability," *Mach. Learn.*, vol. 5, no. 2, pp. 197–227, Jul. 1990. [Online]. Available: http://dx.doi.org/10.1023/A:1022648800760

Bag Fraction	0.75	0.75	0.75	0.75	0.75
Min Leaf Size	1.0	1.0	1.0	1.0	1.0
MaxSplits	32.0	32.0	16.0	64.0	16.0
Min Shrinkage	0.01	0.01	0.01	0.01	0.001
Max Shrinkage	0.1	0.4	0.7	0.4	0.1
RunningTime (seconds)	413.4511	377.9108	292.2713	522.1357	1364.3959
ATD Test RMSE	2.9906	3.0471	3.0163	2.9712	2.9929
ABT Test RMSE	2.9854	3.0449	2.9916	2.9672	2.9657
Cross Validation RMSE	2.9862	3.006	3.0362	3.0383	3.0383
Optimal Num. of Trees	10259	8888	14138.6	5088.4	129227.6
Avg. Leaf Size	66.21538	66.21538	126.076765	33.36434	124.127897
Std. Dev. Leaf Size	0	0	9.721249	0	21.285523
Avg. Splits	32	32	15.4667	64	15.2276
Std. Dev. Splits	0	0	1.1926	0	2.6112
Avg. Shrinkage	0.047665	0.170609	0.338802	0.139473	0.05176
Std. Dev. Shrinkage	0.004657	0.01999	0.035525	0.018966	0.005411

TABLE VIII: Variable Shrinkage: Power Plant Parameters with Best Cross Validation RMSE.

Bag Fraction	0.75	0.75	0.5	0.5	0.5
Min Leaf Size	50.0	1.0	1.0	50.0	50.0
MaxSplits	64.0	8.0	16.0	32.0	128.0
Shrinkage	0.01	0.001	0.001	0.0001	0.001
RunningTime (seconds)	804.0786	1614.8035	1221.8599	2382.7478	565.2622
ATD Test RMSE	0.1432	0.1421	0.1375	0.136	0.1349
ABT Test RMSE	0.1431	0.1417	0.1367	0.1361	0.1349
Cross Validation RMSE	0.1333	0.1335	0.1337	0.1343	0.1345
Optimal Num. of Trees	830.8	26488.2	7292.4	92747.2	9680
Avg. Leaf Size	38.168482	18.110589	15.920001	30.781199	29.906396
Std. Dev. Leaf Size	6.997223	15.62332	4.341818	8.759128	10.357147
Avg. Splits	11.6437	2.752	14.08	7.3093	7.1072
Std. Dev. Splits	1.5399	2.374	3.84	2.0806	2.4574

TABLE IX: Constant Shrinkage: Crime Communities Parameters with Best Cross Validation RMSE.

Bag Fraction	0.5	0.5	0.25	0.75	0.5
Min Leaf Size	50.0	1.0	1.0	50.0	50.0
MaxSplits	32.0	32.0	16.0	128.0	4.0
Min Shrinkage	0.0001	0.001	0.0001	0.001	0.01
Max Shrinkage	0.1	0.1	0.1	0.1	0.1
RunningTime (seconds)	464.4778	1443.1898	830.8513	849.6289	309.6907
ATD Test RMSE	0.1379	0.1434	0.143	0.138	0.1431
ABT Test RMSE	0.1379	0.1429	0.1426	0.1378	0.1418
Cross Validation RMSE	0.1337	0.1338	0.1339	0.1342	0.1343
Optimal Num. of Trees	1139.8	1664.8	2018	1073.4	684
Avg. Leaf Size	34.124605	27.549538	9.0303	34.975735	66.33333
Std. Dev. Leaf Size	1.194561	36.410169	0	1.601863	0
Avg. Splits	8.2961	25.4139	16	12.3821	4
Std. Dev. Splits	0.3004	11.4279	0	0.5222	0
Avg. Shrinkage	0.011279	0.028734	0.029102	0.008565	0.036525
Std. Dev. Shrinkage	0.000291	0.005544	0.010263	0.000314	0.004001

TABLE X: Variable Shrinkage: Crime Communities Parameters with Best Cross Validation RMSE.

[2] ——, "The boosting approach to machine learning: An overview," in *Nonlinear Estimation and Classification*, ser. Lecture Notes in Statistics, D. Denison, M. Hansen, C. Holmes, B. Mallick, and B. Yu, Eds. Springer New York, 2003, vol. 171, pp. 149–171.

[3] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *The Annals of Statistics*, vol. 29, no. 5, pp. pp. 1189–1232, 2001. [Online]. Available: http://www.jstor.org/stable/2699986

[4] A. Natekin and A. Knoll, "Gradient boosting machines, a tutorial," *Frontiers in Neurorobotics*, vol. 7, p. 21, 2013. [Online]. Available: http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3885826/

[5] 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, 2008. [Online]. Available: http://dx.doi.org/10.1111/j.1365-2656.2008.01390.x

[6] G. Ridgeway, "Generalized boosted models: A guide to the gbm package," *Update*, vol. 1, no. 1, p. 14, 2012.

[7] M. Kearns, "Thoughts on hypothesis boosting," *Unpublished manuscript (Machine Learning class project, December 1988)*, 1988.

[8] M. Kearns and L. G. Valiant, "Crytographic limitations on learning boolean formulae and finite automata," in *Proceedings of the Twenty-first Annual ACM Symposium on Theory of Computing*, ser. STOC '89. New York, NY, USA: ACM, 1989, pp. 433–444. [Online]. Available: http://doi.acm.org/10.1145/73007.73049

[9] H. Drucker, R. E. Schapire, and P. Simard, "Improving performance in neural networks using a boosting algorithm," in *Advances in Neural Information Processing Systems 5, [NIPS Conference]*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993, pp. 42–49. [Online]. Available: http://dl.acm.org/citation.cfm?id=645753.668055

[10] Y. Freund and R. Schapire, "A desicion-theoretic generalization of on-line learning and an application to boosting," in *Computational Learning Theory*, ser. Lecture Notes in Computer Science, P. Vitnyi, Ed. Springer Berlin Heidelberg, 1995, vol. 904, pp. 23–37. [Online]. Available: http://dx.doi.org/10.1007/3\-540\-59119-2_166

[11] J. Friedman, T. Hastie, and R. Tibshirani, "Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors)," *Ann. Statist.*, vol. 28, no. 2, pp. 337–407, 04 2000. [Online]. Available: http://dx.doi.org/10.1214/aos/1016218223

[12] J. H. Friedman, "Stochastic gradient boosting," *Computational Statistics And Data Analysis*, vol. 38, no. 4, pp. pp 367–378, 2002. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167947301000652

[13] G. De'Ath, "Boosted trees for ecological modeling and prediction," *Ecology*, vol. 88, no. 1, pp. 243–251, 2007.

[14] J. Elith and J. Leathwick, "Boosted regression trees for ecological modeling," 2015.

[15] "Jsgbm." [Online]. Available: https://code.google.com/p/jsgbm/

[16] "Project implementation." [Online]. Available: https://github.com/ambarket/GBMWithVariableShrinkage

[17] M. Lichman, "UCI machine learning repository," 2013. [Online]. Available: http://archive.ics.uci.edu/ml

[18] "Combined cycle power plant data set." [Online]. Available: http://archive.ics.uci.edu/ml/datasets/Combined+Cycle+Power+Plant

[19] P. Tfekci, "Prediction of full load electrical power output of a base load operated combined cycle power plant using machine learning methods," *International Journal of Electrical Power & Energy Systems*, pp. 126–140, 2014. [Online]. Available: http://dx.doi.org/10.1016/j.ijepes.2014.02.027

[20] P. T. Heysem Kaya, "Local and global learning methods for predicting power of a combined gas & steam turbine," *roceedings of the International Conference on Emerging Trends in Computer and Electronics Engineering ICETCEE 2012*, pp. 13–18, 2012.

[21] "Airfoil self-noise data set." [Online]. Available: http://archive.ics.uci.edu/ml/datasets/Airfoil+Self-Noise

[22] D. P. T.F. Brooks and A. Marcolini, "Airfoil self-noise and prediction," *Technical report, NASA RP-1218*, July 1989.

[23] K. Lau, "A neural networks approach for aerofoil noise prediction," 2006.

[24] R. Lopez, "Neural networks for variational problems in engineering," 2008.

[25] "Bike sharing dataset data set." [Online]. Available: http://archive.ics.uci.edu/ml/datasets/Bike+Sharing+Dataset

[26] H. Fanaee-T and J. Gama, "Event labeling combining ensemble detectors and background knowledge," *Progress in Artificial Intelligence*, pp. 1–15, 2013. [Online]. Available: [WebLink]

[27] "Communities and crime data set." [Online]. Available: https://archive.ics.uci.edu/ml/datasets/Communities+and+Crime

[28] P. T. Heysem Kaya, "Local and global learning methods for predicting power of a combined gas & steam turbine," *roceedings of the International Conference on Emerging Trends in Computer and Electronics Engineering ICETCEE 2012*, pp. 13–18, 2012.

[29] "U.s. department of commerce, bureau of the census producer, washington, dc and inter-university consortium for political and social research ann arbor, michigan. (1992)."

[30] "U.s. department of justice, bureau of justice statistics, law enforcement management and administrative statistics (computer file) u.s. department of commerce, bureau of the census producer, washington, dc and inter-university consortium for political and social research ann arbor, michigan. (1992)."

[31] "U.s. department of justice, federal bureau of investigation, crime in the united states (computer file) (1995)."

[32] M. A. Redmond and A. Baveja, “A data-driven software tool for enabling cooperative information sharing among police departments,” *European Journal of Operational Research 141*, pp. 660–678, 2002.