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*—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. Results show that the use of variable shrinkage results in models with competitive error to those built using constant shrinkage. On average the use of variable shrinkage leads to a considerable decrease in running time and increased robustness to variations in the model parameters.**

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 research presents and evaluates a new way to think about shrinkage for the common case where the base learners are regression trees. We compared our proposed variable shrinkage scheme against the traditional constant shrinkage scheme using four real world datasets with natural regression tasks. Our results show that the use of variable shrinkage results in models with competitive error to those built using constant shrinkage. On average the use of variable shrinkage leads to a considerable decrease in running time and increased robustness to variations in the model parameters without sacrificing resilience to over fitting.

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 $F : \vec{x} \to y$ 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: $bf \in [0, 1]$
        Shrinkage: $v \in [0, 1]$
        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 $\hat{F}_{m-1}(\vec{x})$.
        $g_{m,i} = -\frac{\partial}{\partial \hat{F}_{m-1}(\vec{x_i})} \Psi(y_i, \hat{F}_{m-1}(\vec{x_i})),\ \vec{x}_i \in S_m$    (3)
    Train a new base learner $h_m(\vec{x})$ to predict $\vec{g}_m$ and fit the
    least squares.
        $\beta_m, h_m(\vec{x}) = argmin_{\beta, h(\vec{x})} \sum_{\vec{x}_i \in S_m} [g_{m,i} - \beta h(\vec{x_i})]^2$    (4)
    Solve for the optimal coefficient $\rho$ that minimizes $\Psi$.
        $p_m = argmin_\rho \sum_{\vec{x}_i \in S_m} \Psi(y_i, \hat{F}_{m-1}(\vec{x_i}) + \rho h(\vec{x_i}))$    (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

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

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

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

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

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].

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 (Power Plant)*

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 (Bike Sharing)*

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 (Crime Communities)*

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].

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 I: All Tested GBM Model Parameters

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 II: 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 a myriad of different parameter combinations were tested on the four datasets described in Section VII.

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 I. 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.

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 II for quick reference.

In table III we have the processor model and amount of memory of the systems used to run all of our tests. In addition table III lists the execution limits imposed on each individual test. Specifically, 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.

For a single run on a given dataset, all 2048 parameters were evaluated using the same 80% training set / 20% test set split. Selection of the test set examples was performed uniformly at random. The exact procedure used to evaluate a single set of parameters is provided in algorithm 3.

We performed two full runs for each dataset and computed the averages of data collected for each set of parameters. The results found in section IX and discussed in X are derived from this averaged run data.

Operating System	Ubuntu 14.04.3 LTS
CPU	Intel i5-3570
System Memory	32 GB
Maximum Running Time	1.5 hours
Maximum Memory Usage	20 GB

TABLE III: CPU and Memory Hardware and Limits

1) Use algorithm 2 to find the optimal number of trees.
2) Simultaneously train a GBM using the entire training set (All Training Data GBM), 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 running time, optimal number of trees, and the root mean squared error values at the optimal number of trees for...
    a) The cross validation error (CV RMSE)
    b) The generalization error of the All Training Data GBM (ATD RMSE)
    c) The generalization error of the Aggregated Boosted Tree (ABT RMSE)

Algorithm 3: Procedure for a Single Test.

IX. Results

In tables IV, V, VI, and VII we provide the best constant and variable shrinkage parameters for the Power Plant, NASA, Bike Sharing, and Crime Communities datasets respectively. Each table consists of three subtables. The top subtable summarizes the parameters that achieved the best cross validation error, in the middle you will find the parameters that lead to the All Training Data GBM with minimal generalization error, and finally the bottom subtable presents the parameters that optimize the generalization error of their resulting Aggregated Boosted Tree. Note in an ideal world these would be equivalent, however we can never perfectly estimate generalization error.

For each set of parameters found in these tables, five measured attributes are provided that will be used throughout our discussion to compare performance achieved with constant vs. variable shrinkages. These will be referred to as ”comparison attributes” and consist of the running time, the optimal number of trees, cross validation error, and generalization error of both the All Training Data GBMs and Aggregated Boosted Trees. Note all values for these attributes are averages taken across the two runs performed. For each of these attributes, the cell corresponding to the minimum value is displayed in bold to quickly identify whether constant or variable shrinkage performed better in that particular aspect. In addition, the final column of each table provides the percent difference between the performance of the best constant and best variable parameters. Since we'd like to minimize all of these quantities, a positive percent difference indicates that the best variable shrinkage parameters performed better than their constant shrinkage counterparts.

In order to uncover the overall impact of our proposed shrinkage adaptation scheme, we have generated plots to help visualize how each of the model parameters affects each comparison attribute when used with both constant and variable shrinkage, these will be referred to as ”pairwise parameter influence plots” throughout the remainder of our discussion. The legend for the pairwise parameter influence plots is provided in figure 2 and the plots themselves are found in figures 4, 5, and 6.

Each pairwise parameter influence plot shows the relationship between one of the GBM model parameters (constant shrinkage, min shrinkage, max shrinkage, bag fraction, minimum leaf size, or max number of splits) and one of 3 comparison attributes (running time, optimal number of trees, or cross validation error). The plots for the generalization error of the All Training Data GBMs and Aggregated Boosted Trees have been left out since they are almost exactly the same as those for the cross validation error. This is to be expected of course since we use cross validation to estimate generalization error.

Parameters with lowest CV RMSE			**% Decrease**
Shrinkage Type	Constant	Variable	-
Min Shrinkage	-	0.01	-
Max Shrinkage	0.001	1	-
Bag Fraction	0.75	0.75	-
Min Leaf Size	1.0	1.0	-
MaxSplits	16.0	32.0	-
RunningTime (seconds)	1698.6562	174.0979	**89.75**
Optimal Num. of Trees	149995.5	8432.5	**94.38**
Cross Validation RMSE	2.9382	2.9496	-0.39
ATD Test RMSE	3.0307	3.0948	-2.12
ABT Test RMSE	3.0262	3.0782	-1.72

Parameters with lowest ATD Test RMSE			**% Decrease**
Shrinkage Type	Constant	Variable	-
Min Shrinkage	-	0.01	-
Max Shrinkage	0.001	0.1	-
Bag Fraction	1	0.75	-
Min Leaf Size	1.0	1.0	-
MaxSplits	64.0	64.0	-
RunningTime (seconds)	1394.7333	194.7763	**86.03**
Optimal Num. of Trees	46534.5	5097.5	**89.05**
Cross Validation RMSE	3.0883	3.0531	**1.14**
ATD Test RMSE	2.6946	2.7273	-1.21
ABT Test RMSE	2.7223	2.7145	**0.29**

Parameters with lowest ABT Test RMSE			**% Decrease**
Shrinkage Type	Constant	Variable	-
Min Shrinkage	-	0.01	-
Max Shrinkage	0.001	0.1	-
Bag Fraction	1	0.75	-
Min Leaf Size	1.0	1.0	-
MaxSplits	64.0	4.0	-
RunningTime (seconds)	1394.7333	248.8452	**82.16**
Optimal Num. of Trees	46534.5	46567.5	-0.07
Cross Validation RMSE	3.0883	3.2368	-4.81
ATD Test RMSE	2.6946	2.7378	-1.6
ABT Test RMSE	2.7223	2.7233	-0.04

TABLE IV: Power Plant: Optimal Parameters

We will however provide some additional plots that show the relationship between cross validation error and the two forms of generalization error to see if the variable shrinkages have an impact on the generalization error estimation quality. These are found in figure 3.

To build these plots we first found the average comparison attributes across all datasets and all runs for all sets of parameters, resulting in a single set of 2048 records. Since we are comparing across datasets, 0-1 normalization is first performed on the comparison attributes within each dataset, then the normalized versions are averaged to create the final records used to generate in the plots. As a result, all of the pairwise parameter influence plots have y values ranging from 0 to 1 rather than the original ranges of the plotted attribute.

For each parameter and comparison attribute pair, two plots are generated. In the left columns of figures 4, 5, and 6 you'll find scatter plots with one point for each of the 2048 sets of parameters tested, where the red points indicate tests using constant shrinkage and the blue points correspond to tests using variable shrinkage. To the right of each scatter plot you'll find a line plot where each (x, y) point is the average of all the x values at that particular y value in the scatter plot. Together they provide quite an informative view of how our variable shrinkage scheme affects gradient boosted regression trees overall, and will provide the basis of much of our discussion in section X.

Parameters with lowest CV RMSE			% Decrease
Shrinkage Type	Constant	Variable	-
Min Shrinkage	-	0.01	-
Max Shrinkage	0.01	0.1	-
Bag Fraction	0.25	0.5	-
Min Leaf Size	1.0	1.0	-
MaxSplits	16.0	64.0	-
RunningTime (seconds)	107.2299	494.1639	-360.85
Optimal Num. of Trees	76700	109976	-43.38
Cross Validation RMSE	1.3438	1.3405	**0.25**
ATD Test RMSE	1.293	1.1928	**7.75**
ABT Test RMSE	1.3916	1.278	**8.16**

Parameters with lowest ATD Test RMSE			% Decrease
Shrinkage Type	Constant	Variable	-
Min Shrinkage	-	0.01	-
Max Shrinkage	0.01	0.7	-
Bag Fraction	0.5	0.75	-
Min Leaf Size	1.0	1.0	-
MaxSplits	32.0	16.0	-
RunningTime (seconds)	424.8114	270.0223	**36.44**
Optimal Num. of Trees	149982.5	149634.5	**0.23**
Cross Validation RMSE	1.3641	1.4273	-4.63
ATD Test RMSE	1.1111	1.0764	**3.12**
ABT Test RMSE	1.1653	1.1547	**0.91**

Parameters with lowest ABT Test RMSE			% Decrease
Shrinkage Type	Constant	Variable	-
Min Shrinkage	-	0.001	-
Max Shrinkage	0.01	0.4	-
Bag Fraction	0.5	0.75	-
Min Leaf Size	1.0	1.0	-
MaxSplits	32.0	16.0	-
RunningTime (seconds)	424.8114	222.8172	**47.55**
Optimal Num. of Trees	149982.5	126058	**15.95**
Cross Validation RMSE	1.3641	1.3734	-0.68
ATD Test RMSE	1.1111	1.0881	**2.07**
ABT Test RMSE	1.1653	1.1488	**1.42**

TABLE V: Nasa Air Foil: Optimal Parameters

Parameters with lowest CV RMSE			% Decrease
Shrinkage Type	Constant	Variable	-
Min Shrinkage	-	0.01	-
Max Shrinkage	0.01	0.4	-
Bag Fraction	0.25	0.5	-
Min Leaf Size	1.0	1.0	-
MaxSplits	4.0	32.0	-
RunningTime (seconds)	2.5521	11.1563	-337.14
Optimal Num. of Trees	1934.5	346	**82.11**
Cross Validation RMSE	605.7391	607.1392	-0.23
ATD Test RMSE	754.0447	721.9072	**4.26**
ABT Test RMSE	758.59	717.9233	**5.36**

Parameters with lowest ATD Test RMSE			% Decrease
Shrinkage Type	Constant	Variable	-
Min Shrinkage	-	0.01	-
Max Shrinkage	0.001	0.4	-
Bag Fraction	0.5	0.5	-
Min Leaf Size	1.0	1.0	-
MaxSplits	32.0	128.0	-
RunningTime (seconds)	78.0296	41.6871	**46.58**
Optimal Num. of Trees	12527	862	**93.12**
Cross Validation RMSE	671.2077	694.3465	-3.45
ATD Test RMSE	497.4587	499.4287	-0.4
ABT Test RMSE	498.8692	501.8105	-0.59

Parameters with lowest ABT Test RMSE			% Decrease
Shrinkage Type	Constant	Variable	-
Min Shrinkage	-	0.0001	-
Max Shrinkage	0.01	1	-
Bag Fraction	0.25	0.75	-
Min Leaf Size	1.0	10.0	-
MaxSplits	32.0	8.0	-
RunningTime (seconds)	11.7752	3.1599	**73.16**
Optimal Num. of Trees	915	197	**78.47**
Cross Validation RMSE	663.238	679.7884	-2.5
ATD Test RMSE	502.4053	529.2733	-5.35
ABT Test RMSE	498.5567	502.017	-0.69

TABLE VI: Bike Sharing By Day: Optimal Parameters

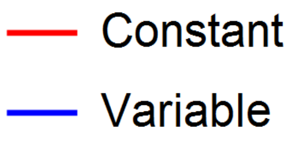

Fig. 2: Plot Legend for Pairwise Parameter Influence Plots

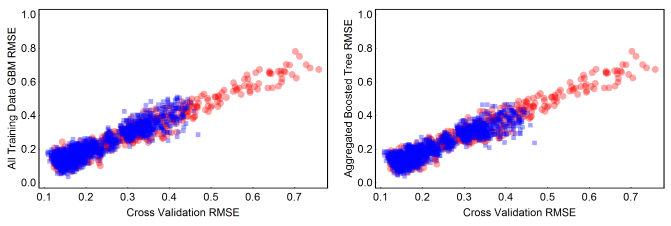

Fig. 3: Cross Validation Error vs. Generalization Error

X. Discussion

Overall the results show that our initial hypothesis was well placed, and there are clear advantages to adapting the shrinkage applied to a leaf node's prediction based on the number of training examples that fell into that node.

From the data shown in tables IV, V, VI, and VII, we can safely say that using variable learning rates does not impede the algorithm's ability to learn and generalize effectively, and in some cases actually results in lower RMSE than using constant shrinkage. However, its important to note that the differences between the error values for the best constant and variable parameters on all datasets are not large enough (only within a 6% difference in all cases) to be considered very significant. So although the variable learning rates show lower minimum errors in all cases for the NASA as well as the Crime Communities dataset, but usually higher minimum errors for the Power Plant and Bike Sharing datasets; the more important result is that the minimum errors achieved with variable shrinkage are comparable in magnitude to those found with constant shrinkage, and not by any means significantly worse.

The really exciting result is the decrease in the amount of time and number of trees it usually takes to reach minimal error. For the Power Plant dataset we see an approximate 85% reduction in running time across the board; and a corresponding reduction in the number of trees required for both the minimum CV Error and ADT Error. In the case of the Aggregated Boosted Trees, the best generalization was achieved using about the same number of trees with both constant and variable shrinkage, but each tree used only 4 splits rather than 64 with constant shrinkage, so here we are still looking at an overall decrease in modeled interactions of about 8 times. We see similar speedups and reduction in modeled interactions across the other three datasets, with only a couple exceptions, namely the minimal cross validation error on the NASA and Bike Sharing datasets; yet on theses datasets we still see between a 35% and 80% decrease in the running time needed to get the achieve the minimal ATD and ABT generalization error. For the Crime Communities dataset, the running time to achieve the minimum error values is reduced by more than 20% across the board.

Interestingly, figure 3, which plots the relationship between cross validation error and generalization errors clearly shows that on

Parameters with lowest CV RMSE			% Decrease
Shrinkage Type	Constant	Variable	-
Min Shrinkage	-	0.0001	-
Max Shrinkage	0.0001	0.1	-
Bag Fraction	0.5	0.75	-
Min Leaf Size	10.0	10.0	-
MaxSplits	64.0	128.0	-
RunningTime (seconds)	3691.4974	696.785	**81.12**
Optimal Num. of Trees	41848.5	3097.5	**92.60**
Cross Validation RMSE	0.1315	0.1306	**0.68**
ATD Test RMSE	0.1418	0.148	-4.37
ABT Test RMSE	0.1416	0.1475	-4.17

Parameters with lowest ATD Test RMSE			% Decrease
Shrinkage Type	Constant	Variable	-
Min Shrinkage	-	0.01	-
Max Shrinkage	0.0001	1	-
Bag Fraction	0.5	0.5	-
Min Leaf Size	10.0	1.0	-
MaxSplits	16.0	64.0	-
RunningTime (seconds)	2538.2906	1983.5935	**21.85**
Optimal Num. of Trees	49239.5	213	**99.57**
Cross Validation RMSE	0.1363	0.1406	-3.15
ATD Test RMSE	0.1239	0.1208	**2.50**
ABT Test RMSE	0.1239	0.1206	**2.66**

Parameters with lowest ABT Test RMSE			% Decrease
Shrinkage Type	Constant	Variable	-
Min Shrinkage	-	0.01	-
Max Shrinkage	0.1	0.7	-
Bag Fraction	1	0.5	-
Min Leaf Size	10.0	1.0	-
MaxSplits	32.0	16.0	-
RunningTime (seconds)	317.9409	80.067	**74.82**
Optimal Num. of Trees	42	147.5	-251.19
Cross Validation RMSE	0.1415	0.1398	**1.20**
ATD Test RMSE	0.1251	0.1222	**2.32**
ABT Test RMSE	0.1238	0.1204	**2.75**

TABLE VII: Crime Communities: Optimal Parameters

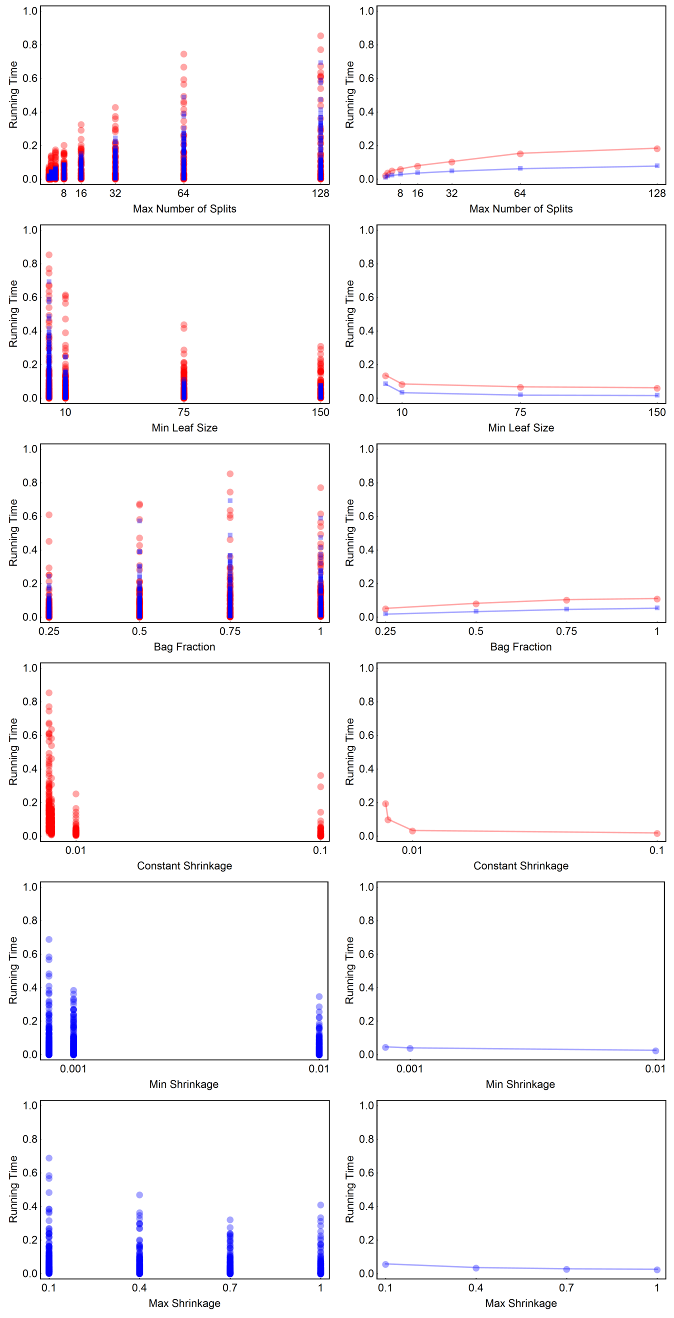

Fig. 4: Parameters vs. Running Time

average the worst case parameters using variable shrinkage result in nearly half the error of the worst case constant shrinkage parameters. This is an exciting result which indicates that variable shrinkage leads to far better model stability in general and is extremely robust to changes in the model parameters. Such a result has obvious real world significance in that model builders could spend less time parameter tuning and still have confidence that their model will perform reasonably well. Looking at the individual parameters in turn, this overall robustness seems to have an intuitive justification.

First consider the case where the GBM is given only a small number of splits it can work with. As always, the regression tree splitting algorithm will make the absolute most out of those splits, choosing at each step the split that results in the largest decrease in squared error (or largest increase in some other measure of node purity). Simple observation seems to indicate that the very first splits in regression trees are often quite unbalanced in nature, leading to large variance in the leaf sizes; and as the number of splits increases this variance diminishes. These initial, uneven, purity maximizing splits contain a lot of valuable information regarding example noise, outliers, and general generalization value that is ignored by simply applying the same small shrinkage to all nodes. With variable shrinkage the GBM is able to leverage this otherwise lost information to make the most out of every split, leading to faster error convergence on average. As the number of splits increases, and the size of the leave nodes become smaller and more evenly spread, it follows logically that the performance seen using constant and variable shrinkage should converge. This is precisely the relationship verified by the Max Number of Splits vs. Cross Validation Error plots in the first row of figure 6, and also explains how the models using variable shrinkage consistently require less running time and less trees to achieve error convergence as seen in the first rows of figures 4 and 5.

We can take a similar view of the minimum leaf size parameter. First observe from the second row of figure 4 that as the minimum leaf size increases, the average running time slowly descends at about the same rate for both constant and variable shrinkages. This is as expected due to the strong correlation between the actual number of splits made and the minimum leaf size. Larger minima result in less possible splits which naturally will trim the running time of the algorithm. From figure 4 we additionally see that the tests using variable shrinkage consistently take about half the time of

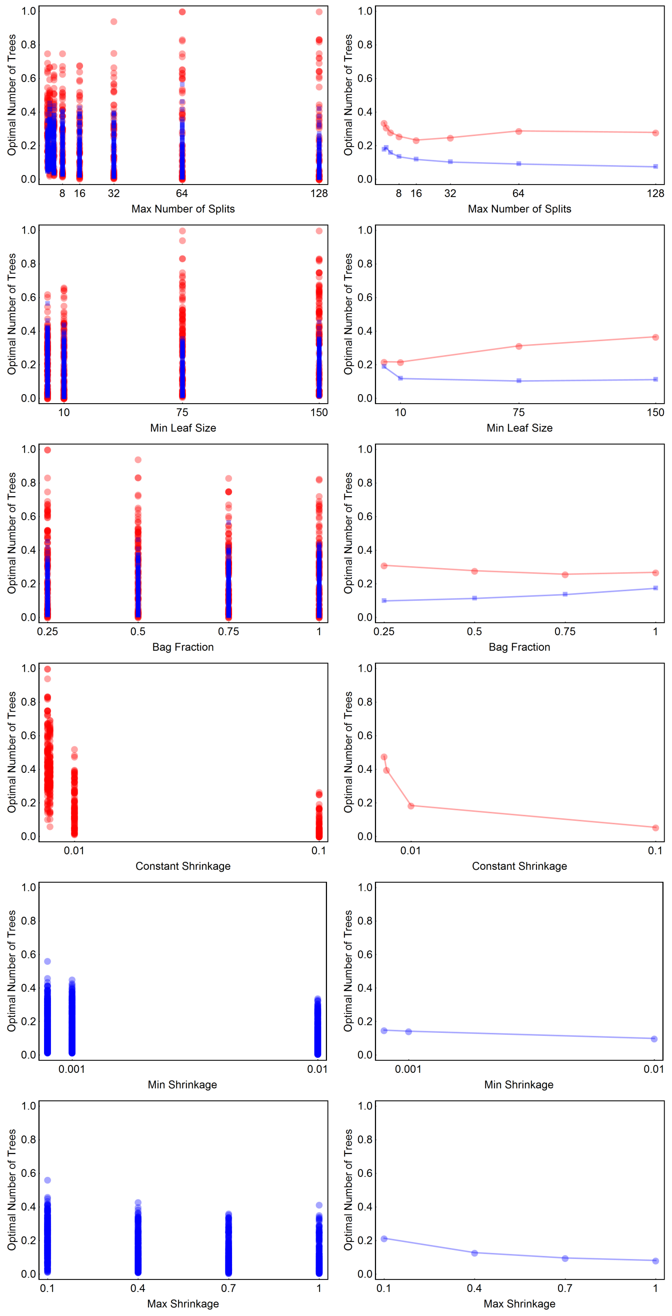

Fig. 5: Parameters vs. Optimal Number of Trees

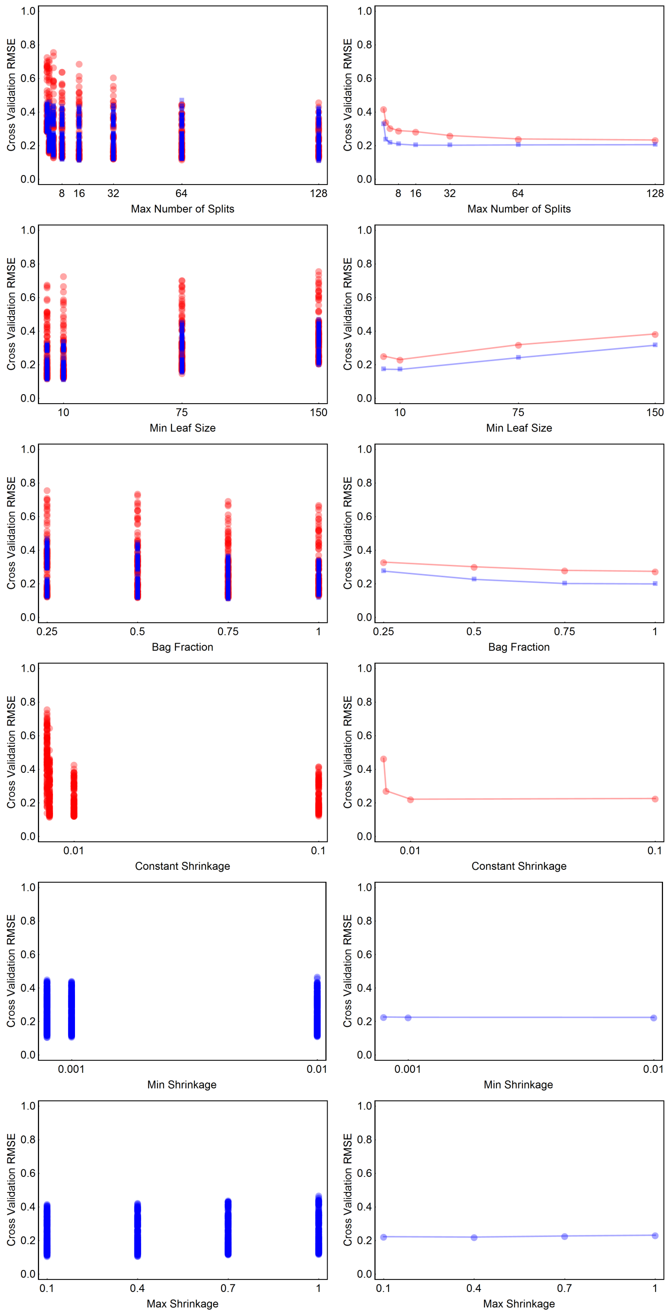

Fig. 6: Parameters vs. Cross Validation RMSE

their constant shrinkage counterparts. Again this is likely due to the fact that variable shrinkages allow the model to make use of more information contained in each split which leads to faster reduction of the overall error and a more timely meeting of the stopping condition. The correlation between leaf size and the actual number of splits becomes more apparent when looking at its relationship to the optimal number of trees and error shown in the second rows of figures 5 and 6. As the leaf size increases, the difference in the number of trees required to converge becomes more pronounced because the computed variable shrinkages will increase as the node sizes increase,; which implies faster learning and stopping condition satisfaction. Although one might expect the use of these larger learning rates to result in a steeper error ascent due to increased risk of over fitting, figure 6 shows that the variable scheme keeps the model stable. Although the error rises as the leaf sizes increase, it rises with a gradient nearly identical to that of the constant tests, indicating once again that the implemented shrinkage weighting scheme enables the GBM to effectively utilize larger learning rates safely without increasing the risk of over fitting.

The bag fraction is a bit more tricky to analyze due to its inherent stochastic nature. When looking at the graphs for each individual dataset there seemed to be quite a bit of variance in which parameter value led to the fastest running time, lowest optimal number of trees, and optimal error; however these variations do appear to have averaged out to some observable trends shown in the third rows of figures 4, 5, and 6. Firstly its once again apparent from figure 4 that variable shrinkage leafs to faster achievement of the stopping condition, less trees, and lower error on average. The running time curve expectedly shows that as the bag fraction increases, the running time increases on average; which makes sense due to the fact that more examples must be evaluated for splitting in the regression trees. The optimal number of trees curve shows an interesting trend in that the constant and variable shrinkage tests seem to be converging on the number of trees required as the bag fraction increases. This could be a due to a decrease in the actual number of splits possible and potentially an increased likelihood of uneven splits when the bag fraction is low and less data is available to each regression tree. Once again the variable shrinkage scheme allows the GBM to make more effective use of all the split information so it requires less trees overall. As the bag fraction increases, the regression tree can make more splits leading to more even sized leaf nodes and an increased resemblance between the computed variable learning rates and the constant learning rates. Finally, in figure 6 we see a slight decrease in average error as the bag fraction increases, yet from the scatter plot we see that the actual minimum errors are achieved using bag fractions less that one. This also follows closely with the intuition that the more training examples you have, the better your predictions will be in general, but is also supportive of Friedman's original reasoning in [12] that providing randomly selected subsets of the training data to each base learner decreases overall model variance, and thus error, by reducing correlation between the predictions made by the base learners built in each successive iteration.

The pairwise parameter influence plots for constant shrinkage are found in the fourth rows of figures 4, 5, and 6. These plots confirm the trends that have been widely analyzed by previous Gradient Boosting Machine research. Increasing the shrinkage leads to less trees and faster stopping condition satisfaction, but at the cost of over fitting and increasing error. In many cases the tests using the two lowest constant shrinkage values of 0.0001 and 0.001 were unable to converge by the time they reached the maximum number of trees or ran out of time or memory, thus the error is actually the highest for these lowest error rates, but surely would have decreased if allowed to run to completion. Also note that the plots do not capture the error increase that will result from too high of constant shrinkage since the highest tested learning rate was 0.1. In early tests larger learning rates were tested and invariably resulted in unstable models and quick error divergence.

We will conclude our discussion with a look at the pairwise parameter influence plots for the min and max shrinkage parameters, which uncover what is perhaps the most surprising result of this study. It seems regardless of the values chosen for these parameters, the distribution of error values across all tests remains unchanged. Smaller values of these parameters do result in more trees and slightly longer running times on average, which makes sense since the average computed learning rates would be smaller and thus the model would learn more slowly. But the error value distribution is simply unaffected. This is great news as it in many ways nullifies the only major downside of implementing and using this scheme, the addition of another parameter to tune. Based on our results, it seems that in practice the min and max shrinkages could simply be set to any reasonable values, then parameter tuning would only need to be done on the remaining parameters.

Our analysis strongly supports our initial hypothesis that by varying the shrinkage applied to the prediction of the examples in each individual leaf of regression tree base learners, the convergence time can be decreased without sacrificing generalization ability and resilience to over fitting. We have shown that the proposed variable shrinkage scheme consistently achieves comparable root mean squared error to constant shrinkage on a variety of real world datasets. In addition the pairwise parameter influence plots indicate that by dynamically computing shrinkage based on leaf node size the algorithm as a whole becomes significantly more robust to variations in all model parameters.

XI. Conclusion

A variable shrinkage scheme for gradient boosting machines with regression tree base learners was developed and tested against four real world datasets of various sizes, predictor dimensionality, and regression task difficulty. Under the proposed adaption scheme, a separate shrinkage is calculated for each leaf node in each regression tree such that leaf nodes with less training examples receive lower shrinkage than larger sized leaf nodes. In this way less weight is given to the predictions of noisy, outlying, and atypical examples while the model focuses its learning to the responses of the more typical training examples.

Our analysis shows that this adaption scheme leads to a significant decrease in the total number of interactions that must be modeled in order to reach a minimum cross validation error. This decrease in running time and number of trees built is not accompanied by a decreased resilience to over fitting as one might expect, and has been shown to be capable of generating models with competitive and in some cases better root mean squared error vs. the best error achieved using a traditional constant shrinkage scheme.

In addition, GBMs built with variable shrinkage have been shown to be far more robust to variations in all of the model parameters, leading to far lower error with the worst case variable shrinkage parameters than with the worst case constant shrinkage parameters. This property along with the decreased average running time makes our variable shrinkage scheme an attractive candidate for further research and use in production gradient boosting machine implementations.

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

[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.